Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: **Розроблення програмного забезпечення схем відновлення для розподілених обчислень**

Виконав: студент VI курсу, групи КН-61м
спеціальності
122 – "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Буца А.Б.
(прізвище та ініціали)
Керівник Різник О.Я.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Львів – 2024 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Борецька І. Б.

"____"_____2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Буца Андрій Богданович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розроблення програмного забезпечення схем відновлення для розподілених обчислень**

керівник роботи, Різник Олег Яремович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 13.02.2023 року № С-49

2. Термін подання студентом роботи 05. 01. 2024 р.

3. Вихідні дані до роботи:

- провести огляд алгоритмів синтезу схем відновлення для розподілени обчислень;
- дослідити математичну модель синтезу схем відновлення для розподілени обчислень;
- розробити програмний продукт з інтуїтивним інтерфейсом;
- провести тестування роботи розробленого програмного продукту.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
Висновки

5. Перелік графічного матеріалу:

системний аналіз, розробка та відображення алгоритму роботи схем відновлення для розподілени обчислень, структура програмного рішення, тестування, знімки екранів із відображенням процесу роботи програмного продукту
Додаток. Вихідний код розробленого програмного забезпечення

6. Дата видачі завдання 15 лютого 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання	15.02.2023	виконано
2	Огляд літературних джерел	30.04.2023	виконано
3	Розділ 1. Стан проблемної області	31.07.2023	виконано
4	Розділ 2. Інформаційне забезпечення	30.08..2023	виконано
5	Розділ 3. Математичне забезпечення	29.09.2023	виконано
6	Розділ 4. Програмне забезпечення	31.10.2023	виконано
7	Розділ 5. Розробка стартап-проєкту	30.11.2023	виконано
8	Оформлення пояснювальної записки	29.12.2023	виконано
9	Подання готової роботи	05.01.2024	виконано

Студент _____ Буца А.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Різник О.Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістеоська робота містить 56 сторінок пояснювальної записки, 7 рисунків, 4 таблиqs, 3 додаткf, 31 джерело.

В даній роботв розроблені схеми відновлення розподілених обчислень на основі лінійок Голомба, і в результаті були знайдені обмеження їх використання до 492 комп'ютерів у розподіленій системі. В результаті дослідження отримано результати побудови ліній Голомба в програмі Visual Prolog до порядку 24.

Ключові слова: *лінійка Голомба, схема відновлення, розподілене обчислення, кластер*.

ANNOTATION

Magisterial work contains 56 pages of explanatory note, 7 figures, 4 tables, 3 appendices, 31 sources.

In this work, schemes for restoring distributed calculations based on Golomb lines were developed, and as a result, limitations of their use up to 492 computers in a distributed system were found. As a result of the study, the results of constructing Golomb lines in the Visual Prolog program up to order 24 were obtained.

Keywords: *Golomb ruler, chart of renewal, up-diffused calculation, cluster*.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне та алгоритмічне забезпечення синтезу схем відновлення для розподілених обчислень, а саме:

1. провести попередній аналіз існуючих схем відновлення для розподілених обчислень;
2. визначити математичну модель схем відновлення для розподілених обчислень;
3. визначити розподіл навантаження схем відновлення для розподілених обчислень;
4. розробити алгоритм синтезу схем відновлення для розподілених обчислень на вибраній математичній моделі;
5. провести інтерпретацію отриманих результатів;
6. розробити програмне забезпечення для представлення результатів роботи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ОДИНИЦЬ І ТЕРМІНІВ

ЛГ	- лінійка Голомба
СВГ	- схема відновлення Голомба
ОСВ	- ощадлива схема відновлення
МСВ	- модульна схема відновлення
OGR	- оптимальні лінійки Голомба

ВСТУП

Актуальність дослідження. Одним із способів досягнення високої незалежності від помилок є запуск програми в кластері або розподіленій системі. Є головний комп’ютер, на якому програма працює в нормальних умовах, і допоміжний комп’ютер, який виконує завдання, коли головний комп’ютер вимкнено. Також може бути третій комп’ютер, який виконуватиме завдання, коли основний комп’ютер і додатковий комп’ютер перебувають у режимі офлайн тощо.

Метою дослідження є вивчення закономірностей схем відновлення. Для досягнення ефективного рішення необхідно знайти схему відновлення, що працює в режимі реального часу.

Предметом дослідження є схеми відновлення для розподілених обчислень.

Об'єктом дослідження є алгоритм синтезу схем відновлення для розподілених обчислень.

Схема відновлення повинна мати найбільш оптимальне навантаження навіть при найбільш несприятливих поєднаннях комп'ютерів, що вийшли з ладу, тобто. Найгірший випадок має бути оптимізований. Ще однією перевагою використання розподілених систем або кластерів, окрім відмовостійкості, є розподіл навантаження між комп’ютерами. Коли всі комп’ютери працюють, хочемо рівномірно розподілити навантаження. Однак навантаження на деякі комп’ютери збільшиться, коли один або кілька комп’ютерів вийдуть з мережі, але навіть за цих умов навантаження на інші комп’ютери може бути рівномірним.

Практична цінність. Кластери та розподілені системи стійкі до збоїв і характеризуються високою продуктивністю завдяки розподілу навантаження. Коли всі комп’ютери запущені, хочемо рівномірно розподілити навантаження між комп’ютерами. Якщо один або кілька комп’ютерів

виходять з ладу, навантаження на ці комп’ютери має бути розподілено між іншими комп’ютерами в кластері. Перерозподіл визначається схемою оновлення. Схема оновлення контролюється послідовністю цілих чисел за модулем *n*. Кожна послідовність гарантує мінімальне навантаження на комп’ютер, який має максимальне навантаження, навіть якщо найгірші комбінації комп’ютерів вимкнено. Використовуючи грубу силу, ми обчислюємо найкращі можливі шаблони оновлення для будь-якої кількості несправних комп’ютерів. Проте пошук дуже складний. Оптимальні послідовності, а отже, оптимальна межа передачі, представлені для максимум двадцяти одного комп’ютера в розподіленій системі або кластері.

Розподіл навантаження при вимкненому комп'ютері вирішується шляхом оновлення списків процесів, запущених на пошкодженому комп'ютері. Набір усіх списків оновлення називається схемою оновлення. Тому розподіл навантаження повністю визначається схемою відновлення для будь-якої кількості вимкнених комп'ютерів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Характеристика об’єкту дослідження

Припускаємо, що робота, виконана приблизно n комп'ютерами, повинна бути передана як одна атомна одиниця. Це типовий сценарій у випадках, коли:

– робота, яку виконує комп'ютер, виконується з однієї зовнішньої системи;
– існує одна мережева адреса для кожного комп’ютера, і використовуємо протокол IP (або подібну технологію);
– уся робота, що виконується комп’ютером, виконується одним процесом або групою пов’язаних процесів, які складають спільні локальні ресурси, і тому повинні бути передані як єдина атомарна одиниця.

Деякі комп'ютерні мережі, такі як гіперкуби, обмежують шляхи зв'язку в системі таким чином, що повідомлення від комп'ютера *A* до комп'ютера *B* повинні в деяких випадках надсилатися через комп'ютер *C* (або *D*). Однак багато систем на основі шини, наприклад ті, що використовують Ethernet [13], використовують прямий зв'язок між будь-якими парами вузлів у кластері [26]. Ми припускаємо, що можна відновити роботу будь-якого комп'ютера в кластері (або, в даному випадку, перенаправити канал зв'язку на зовнішню систему обробки даних кластера).

Розглядаємо систему або кластер як комбінацію з *n* однакових комп'ютерів. Під час нормальної роботи кожен комп’ютер має один процес. Робота розподіляється між ними порівну *n* процесами. Кожен процес має список оновлень, який визначає, на якому комп’ютері процес має перезапуститися, якщо поточний комп’ютер вимкнеться. Якщо комп’ютер виходить з ладу, процес буде перенесено на наступний комп’ютер у списку тощо.

Припускаємо, що процеси відкатуться, як тільки комп'ютер відновить роботу. У найбільших кластерних системах це може бути визначено користувачем, що означає, що в деяких випадках може не виникати необхідності автоматично перерозподіляти процеси, коли повертається непрацюючий комп’ютер.

Процес в комп'ютері i $0 \le i \le n-1$, переміщається в комп'ютер $x_{i,1}$, якщо комп'ютер i вимкнений. Якщо комп'ютер $x_{i,1}$ також вимкнений, або ламається перед тим, як комп'ютер i вимкнеться, той же процес переміщається в комп'ютер $x_{i,2}$, і так далі. Загалом, якщо усі комп'ютери $x_{i,1}$, $x_{i,2}$, …, $x_{i,j-1}$ вимикаються, і комп'ютер $x_{i,j}$ працює, процес, який був спочатку в комп'ютері i знаходиться в комп'ютері $x_{i,j}$.

Рис. 1.1 є прикладом перерозподілу процесу від комп'ютера нуль. Комп'ютери нуль, один і шість вимикаються. Тому процес від комп'ютера нуль на комп'ютері $x_{0,3}=3$.

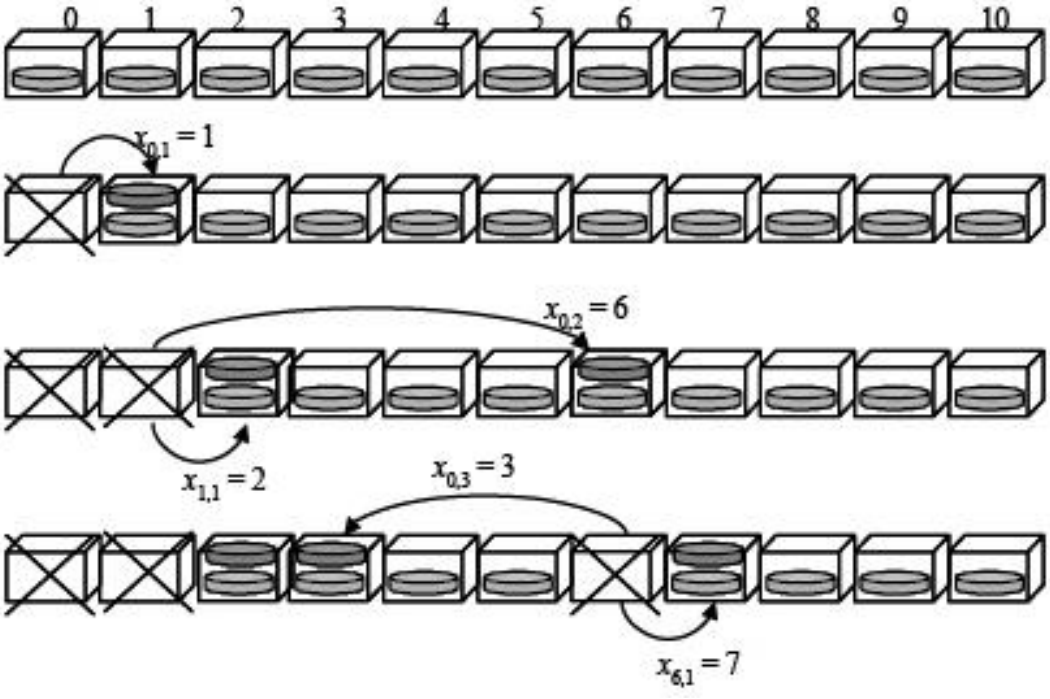


Рис. 1.1. Трансфер процесу від комп’ютера 0

Усі записи в списку оновлення мають бути чітко визначені. В основному працюємо зі схемами регулярного оновлення. Тому всі списки структуровані вектором R, як показано нижче:

$$0{:}<R_1, R_2, \ldots, R_{n-1}>$$

$$1{:}<R_1+1, R_2+1, \ldots, R_{n-1}+1>$$

$$\ldots$$

$$n-1{:}<R_1+n-1, R_2+n-1, \ldots, R_{n-1}+n-1>,$$

де всі числа обчислені за *mod n*.

Позначаємо перший комп'ютер R_0=0 і усі комп'ютери через $x_{0,i}$, R_i для i=1, …, n–1. Відмітимо, що процес спочатку в комп'ютері нуль знаходиться в комп'ютері R_j після j вимкнень.

Наприклад, для n=6 ми маємо такі регулярні шаблони оновлення векторних даних R=<0, 1, 3, 5, 4, 2>:

0: 1, 3, 5, 4, 2

1: 2, 4, 0, 5, 3

2: 3, 5, 1, 0, 4

3: 4, 0, 2, 1, 5

4: 5, 1, 3, 2, 0

5: 0, 2, 4, 3, 1.

Кількість регулярних схем оновлення довжиною n (n комп'ютерів) позначено *R(n)*. Там явно є (n–1)! відмінних таких послідовності з n комп'ютерами, так як $R(n)$=(n–1)!.

1.2. Задача виявлення оптимальної схеми відновлення

Проблему визначення оптимальної (або навіть кращої) схеми оновлення досліджено в [16, 17]. У магістерській роботі хочемо розрахувати найкращу можливу схему відновлення для всіх n, що є метою роботи.

Більше комп’ютерів кластера підтримують цей тип оновлення помилок, наприклад, список вузлів у кластері Sun [17], список пріоритетів

MC/ServiceGuard (HP) [5], каскадна група в HACMP (IBM) [6] і у кластерах Windows Server 2003 (Microsoft, раніше називався MSCS) [22].

У [5] говоримо про відмовостійкість у розподіленій обробці. Балансування навантаження та придатність особливо важливі у відмовостійких розподілених системах, де важко вгадати, на якому комп'ютерному процесі повинні працювати. Це *NP*-повна задача для великої кількості комп'ютерів [30].

Дослідники визначають відмовостійкість як здатність системи негайно реагувати на несподівану апаратну або програмну помилку [18]. Тому багато відмовостійких комп'ютерних систем віддзеркалюють усі дії, наприклад, для кожної дії виконуються дві або більше подвійних систем у тому сенсі, що якщо одна виконує це, інша може взяти на себе її роботу. Ця методика також використовується в кластерах [23].

Цим завданням можна керувати динамічно, коли рішення про переміщення не залежать від поточного стану системи. Проблема динамічної політики досить непередбачувана. В іншому випадку необхідно використовувати статичну політику, на основі якої базується інформація про поведінку системи. У цьому випадку рішення про перенесення не залежить від поточного стану системи. Це робить їх менш складними та більш передбачуваними, ніж динамічні політики [15]. Крім апаратних збоїв, можливі випадкові збої, спричинені програмними подіями.

У [29] представлені та оцінені «дворівневі» схеми відновлення. У кластері реалізована схема оновлення. Автори оцінюють продуктивність і затримку контрольної точки під час виконання схеми оновлення. Для систем, орієнтованих на дії, в [7] Геленб запропонував «багатоконтрольний підхід», тобто подібний до багатогоризонтної схеми відновлення, представленої в [29]. Математичну модель системи, орієнтованої на дію, у разі періодичних відмов запропоновано в [9]. У цьому випадку система може працювати за стандартною схемою відновлення контрольної точки. У [3] і [8] автори

пропонують кілька алгоритмів, які можуть сформулювати збої завдань і перезапустити їх. Вони аналізують поведінку паралельних програм, представлених навколо випадкової діаграми завдань у багатопроцесорному середовищі. Проте всі ці алгоритми працюють динамічно.

Якщо комп'ютер зламано, центр обміну повинен відправити дані на інший комп'ютер у кластері; це означає, що необхідно мати список оновлень, пов’язаний з кожним центром обміну.

Багато засобів запуску кластерів пропонують не лише визначені користувачем списки оновлень, але й схеми динамічного балансування навантаження, коли вузол не працює.

1.3. Попередні дослідження

Робота [20] розглядає проблему пошуку моделі відновлення, яка може гарантувати оптимальний випадок поширення навантаження, коли більшість комп’ютерів виходять з мережі. Схеми повинні мати якомога більше значення *x*. Алгоритм *Log* представлено та перевірено, будуючи схеми відновлення, які гарантують оптимальність із максимально можливою кількістю $log_2 n$ комп'ютерів, коли вони вимкнені. Тут максимальне число процесів на тому ж комп'ютерів після *k* вимкнень − *BV(k)*, де функція *BV(k)* забезпечує нижню границю для будь-якої схеми статичного відновлення. *BV* є визначенням, що зростає і містить точно *k* входів. Це дорівнює *k* для усього *j−того* входу у вектор *BV* і має значення $\left(\sqrt{2(j+1)}+1/2\right)$. Отже *BV*(*p*)=<2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, …>

Два інших алгоритми, економний алгоритм і схема Голомба, розглядаються в [16]. Ці алгоритми створюють схеми відновлення, які забезпечують кращу оптимальність, ніж алгоритм *Log* (тобто краще балансування навантаження). Економний алгоритм ґрунтується на математичній задачі ідентифікації послідовності натуральних чисел такої, що вся сума послідовностей є унікальною та мінімальною. Розрахувати економний алгоритм легко навіть для великих *n*. Список оновлень для

комп’ютера складається з двох частин. Перша частина - це рядок з економного алгоритму, а друга частина заповнена залишковими числами.

Діаграма Голомба є окремим випадком алгоритму *Lean*. Назва цього методу походить від рядка Голомба [2, 6]. Математичне формулювання було змінено на основі найдовшої послідовності натуральних чисел, так що сума послідовності менша або дорівнює n, а всі суми, що містяться в послідовності, різні, де n – кількість комп’ютерів у кластері.

Списки оновлень Голомба формулюються, наприклад, як економні списки оновлень: для k комп’ютерів у кластері створюємо список оновлень для нульового комп’ютера, використовуючи відомий рядок Голомба із сумою, меншою або рівною k, а решта список оновлення заповнюється рештою чисел до $k-1$, наприклад, для $k=12$ маємо список <1, 4, 9, 11, 2, 3, 5, 6, 7, 8, 10>.

Схеми Голомба забезпечують оптимальність для більшої кількості відключених комп'ютерів, ніж економічна схема. Проте пошук (і доказ) оптимальних схем Голомба стає все більш важливим із збільшенням кількості (n) комп’ютерів [32, 33]. Отже, для великих n можемо легко обчислити послідовність із явними частковими сумами на основі економного алгоритму, у якому ще невідомі послідовності Голомба.

У [17] проблему було оптимізовано для циклічного сценарію: процес відправляється назад або проходить через головний комп’ютер. Це відповідає новій математичній задачі ідентифікації найдовшої послідовності додатних цілих чисел, сума всіх послідовностей яких унікальна за модулем n.

Висновки до розділу 1

Це математичне формулювання комп’ютерної задачі забезпечує нові, більш ефективні схеми відновлення, які називаються модульними, оптимально для більшої кількості невдалих схем. комп’ютери в оригінальній комп’ютерній задачі. Це означає, що ці результати відображають поточний стан проблеми.

Тому основною проблемою є розробка ефективного алгоритму синтезу ліній Голомба будь-якого порядку з метою побудови на їх основі схем оновлення.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Визначення навантаження схем відновлення

Визначимо навантаження найбільш завантаженого комп'ютера після збоїв p, використовуючи оптимальні схеми відновлення, вибираючи схеми, що мінімізують це навантаження.

Нехай $L(n, p, J, R)$ $(n>p)$ означають навантаження на найбільш сильно навантажений комп'ютер коли p комп'ютерів $J=\{j_1, \ldots, j_p\}$ вимикаються, і використовується схема відновлення R.

Нехай $L(n, p,R)=\max JL(n, p, J, R)$ для усіх векторів $J=\{j_1, \ldots, j_p\}$, тобто для всіх комбінацій p відключених комп'ютерів $L(n, p, R)$. визначає поведінку у найгіршому випадку після p вимкнень. Розцінюємо $L(n, p, R)$ як послідовність в змінному p і називаємо її послідовністю навантаження.

Наприклад, якщо $p=3$, то маємо: $L(n,\{1, 2, 3\},)=<2, 2, 3>$, для $(n>p)$ і схему відновлення R. Якщо один комп'ютер виходить з ладу, навантаження на найбільш завантажений комп'ютер дорівнює 2, якщо виходять з ладу два комп'ютери, навантаження також дорівнює 2, але якщо виходять з ладу три комп'ютери, навантаження на найзавантаженіший комп'ютер дорівнює 3 (найгірший сценарій).

2.2. Послідовне балансування навантаження

Під час балансування навантаження надаємо чіткий пріоритет невеликій кількості офлайн-комп'ютерів над більшою кількістю. Називаємо це послідовним балансуванням навантаження. Визначаємо набір схем відновлення у формулі, яка мінімізує максимальне навантаження на відключені комп'ютери:

$$R(n, p) = \left\{ R \in R9n) \mid L(n,i,R) - \min_{P \in R(n)} L(n,i,P),1,..,p \right\}$$

Відколи $L(n, 1, R)=2$ для будь-якої схеми R нам потрібно мінімізувати максимальне навантаження на всі схеми, коли лише один комп'ютер

виходить з ладу. Тоді максимальне навантаження на найбільш завантажений комп'ютер дорівнює двом. Коли два комп'ютери виходять з мережі, навантаження на найбільш завантажений комп'ютер може становити два або три. Оскільки шукаємо мінімум схеми, це зміщення дорівнює двом.

Зараз, серед схем в $R(n, 2)$ вибираємо схеми, які мінімізують навантаження на найбільш завантажений комп'ютер, коли три комп'ютери вимкнені для формування набору $R(n, 3)$.

Отже, за визначенням: $R(n, 3) \subset R(n, 2)$. Таким чином, множина $R(n, 3)$ містить схеми відновлення, які мінімізують накладні витрати під час визначення пріоритетів виконання, коли два комп'ютери знаходяться в автономному режимі. Це послідовне балансування навантаження.

Взагалі формуємо $R(n, p+1)$ вибираючи послідовності відновлення у $R(n, p)$, яке має бути мінімальним $L(n, p+1, R)$. Отже $R(n, p)$, не порожне для усього $p=1, \ldots, n-1$ і маємо включення $R(n, p+1) \subset R(n, p)$ для усього $p=1, \ldots, n-1$.

Відмітимо, що там можливо $R \notin R(n,p)$, і існували б схеми відновлення таким чином, що $L(n, p,R)<L(n, p,R')$, $R' \subset R(n, p)$. Потім $L(n, i, R)>L(n, i, R')$ для якогось R' є $R(n, p)$ і $i<p$. Це наслідок послідовного балансування навантаження. Послідовність $L(n, 1, R)$, $L(n, 2, R)$, …, $L(n, n-1, R)$ ідентична для усіх $R \epsilon R(n, n-1)$. Позначимо цю оптимальну послідовність навантаження SV.

В табл. 2.1. нижче наведено кілька прикладів модуля послідовності 6. Їх 5. Можливі наступні послідовності.

Таблиця 2.1.

Приклади послідовностей модуля 6.

Послідовності	p=1	p=2	p=3	p=4	p=5
$R1=\{0, 1, 3, 5, 4, 2\}$	2	2	3	3	6
$R2=\{0, 1, 3, 5, 2, 4\}$	2	2	3	4	6
$R3=\{0, 3, 1, 4, 2, 5\}$	2	2	4	4	6
$R4=\{0, 2, 4, 1, 5, 3\}$	2	3	3	3	6
$R5=\{0, 2, 4, 1, 3, 5\}$	2	3	3	4	6

Ці послідовності дозволені у разі відмови одного комп'ютера (відносяться до $R(6, 1)$). У разі двох пошкоджених комп'ютерів послідовності $R4$ і $R5$ виключаються, решта належить до $R(6, 2)$. У разі трьох відключень $R3$ вимикається, а у разі чотирьох відключень $R2$ вимикається. Як наслідок, оптимальна послідовність у цьому прикладі - $R1$. Якщо спостерігаємо лише послідовності $R3$ і $R4$, то маємо це, хоча $L(6, 4, R4)<L(6, 4, R3)$, оскільки $p=2$, то маємо $L(6,2, R4)>L(6, 2, R3)$.

2.3. Оптимальні схеми відновлення

Послідовності в $R(n, p)$ називають $p-$оптимальними послідовностями.

Визначення 2.1. Множина $R^*(n)=R(n, n–1)$ – сукупність послідовно оптимальних схем відновлення.

Основною метою магістерської роботи є опис та синтез множин для всіх n та їх навантажень у вигляді послідовності SV. Нехай $MV = \max\left(BV(j), \left\lceil \frac{n}{n-y} \right\rceil \right)$. Для усіх $R \in R(n)$, це доводиться в [20] що $MV \leq L(n, j, R)$ хоча $j=1, 2, \ldots, p$.

Ланцюг – послідовність $x_{i,1}, x_{i,2}, \ldots, x_{i,j}$ списку відновлення, який закінчується в комп'ютері y, тобто $y=x_{i,j}$. Це представляє шлях, що *і–тий* процес (процес *і–того* комп'ютера) закінчується в комп'ютері y. Змінна j позначає кількість невдалих кроків до y. Говоримо, що процес i має відстань j

від комп'ютера *y*. Починаючи з запису, для кожного виправленого списку та в кожному списку регулярних оновлень існує рівно один процес із відстанню *j* на весь шлях. Потім розставляємо ланцюжки, збільшуючи відстань до *y*:

$$L_1=\{y_{1,1}\}$$

$$L_2=\{y_{2,1}, y_{2,2}\}$$

Два рядки однакової довжини не можуть існувати, коли процеси народжуються на одному комп'ютері. Підсумовуючи: кожна схема представляє шлях процесу, який передається від комп'ютера до наступного вимкнення комп'ютера. Кожен вхід у схемі представляє комп'ютер, який вимкнено.

Список регулярних оновлень містить рядки:

$\{y–R_1\}$ (mod n)

$\{y–R_2, y–(R_2–R_1)\}$ (mod n)

$\{y–R_3, y–(R_3–R_1), y-(R_3–R_2)\}$ (mod n)

…

$\{y–R_{n–1}, y–(R_{n–1}–R_1), \ldots, y–(R_{n–1}–R_{n–2})\}$ (mod n)

Можемо, альтернативно описати регулярну схему відновлення $R=\{0, R_1, \ldots, R_{n–1}\}$ різницями $r=\{r_1, r_2, \ldots, r_{n–1}\}$.

Визначаємо

$$r_1=R_1 \text{ (mod n)}$$

$$r_2=R_2–R_1 \text{(mod n)}$$

$$r_3=R_3–R_2 \text{(mod n)}$$

…

$$r_{n–1}=R_{n–2}–R_{n–1} \text{(mod n)}$$

і отже

$$R_1=r_1(\text{mod n})$$

$$R_2=r_1+r_2(\text{mod n})$$

…

$$R_{n-1}=r_1+r_2+\ldots+r_{n-1}(\text{mod n})$$

Близькі регулярні схеми відновлення:

0. $\{r_1, r_1+r_2, r_1+r_2+r_3, \ldots, r_1+\ldots+r_{n-1}\}$ (mod n)

1. $\{r_1+1, r_1+r_2+1, \ldots, r_1+\ldots+r_{n-1}+1\}$ (mod n)

…

n-1 $\{r_1+n-1, \ldots, r_1+\ldots+r_{n-1}+n-1\}$ (mod n)

Доступ до комп'ютерних схем, які закінчуються на y, здійснюється наступним чином:

$\{y-r_1\}$ (mod n),

$\{y-(r_1+r2), y-r_2\}$ (mod n),

$\{y-(r_1+r_2+r_3), y-(r_2+r_3), y-r_3\}$ (mod n),

…

$\{y-(r_1+\ldots+r_{n-1}), \ldots, y-(r_{n-2}+r_{n-1}), y-r_{n-1}\}$ (mod n),

Властивості, які нас цікавлять, показані в наступній версії:

$C_1=\{r_1\}$ (mod n),

$C_2=\{r_1+r_2, r_2\}$ (mod n),

$C_3=\{r_1+r_2+r_3, r_2+r_3, r_3\}$ (mod n),

…

$C_{n-1}=\{r_1+\ldots+r_{n-1}, \ldots, r_{n-2}+r_{n-1}, r_{n-1}\}$ (mod n),

Розглянемо приклад послідовності.

Для n=7 регулярного списку відновлень для комп'ютера 0 є: { , $R2$, …, $R6$}={0, 1, 4, 6, 2, 3, 5} , описано відмінності (mod 7) r:

$$\{r_1, r_2, \ldots, r_{n-1}\}=\{1, 3, 2, 3, 1, 2\}.$$

Доступ до комп'ютерних схем, які закінчуються на y, здійснюється наступним чином:

{*y*–1} mod 7, {*y*–4, *y*–3} mod 7, {*y*–6, *y*–5, *y*–2} mod 7, і так далі. Послідовності, що вивчаються є: C_1={1} C_2={4, 3} C_3={6, 5, 2}, і так далі. Отже, намалюємо різниці по діагоналі від низу в трикутнику на рис. 2.1.

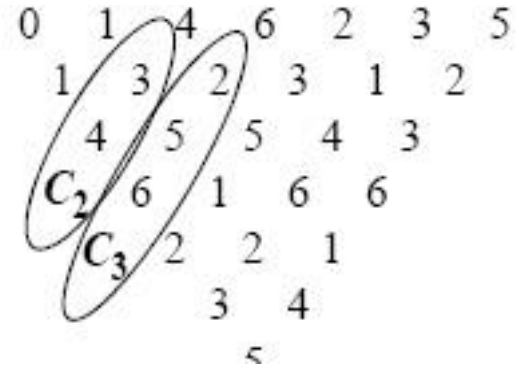


Рис. 2.1. Різницевий трикутник для *n=7*

2.4. Нормальні послідовності

Вимога, щоб входи в *R*={0, R_1, …, R_{n-1}} − чіткі перетворення для відмінностей *r*={r_1, r_2, …, r_{n-1}} у вимозі, йдеться про те що усі послідовні парні суми не дорівнювали нулю (mod *n)*:

Визначення 2.2. Послідовність *r*={r_1, r_2, …, r_{n-1}} нормальна, якщо $\sum_{i=a}^{b} r_i \neq 0(\mathrm{mod} n)$ для усіх 1≤*a*≤*b*≤*n*–1.

Анотація 2.1: *R* має чіткі нормальні входи *r*.

Доказ: Маємо *Rj=Rk* 0≤*k*≤*j*≤*n*–1, тоді і тільки тоді, коли $\sum_{i=1}^{j} r_i = \sum_{i=1}^{k} r_i (\mathrm{mod} n)$

вірно, якщо і тільки тоді якщо $\sum_{i=k+1}^{j} r_i = 0(\mathrm{mod} n)$, то анотація доведена.

Отже, якщо будь-яка послідовна часткова сума − *n*, то *Ri=Rj* для деякого *i≠j*. Простий модуль більший за довжину різницевої послідовності. Наприклад, послідовності 1:*s* є лише нормальними *r*={1, 1, 1, …}, відповідають *R*={0, 1, 2, …}. Відмічено, що (*n*–1)! часткових сум мають різний нормальний стан послідовності довжини *n*–1.

2.5. Приклади балансування навантаженням

Щоб забезпечити оптимальний час балансування навантаження, нам потрібно якомога більше компоненті $\sum_{i=a}^{b} r_i$ не лише не рівних нулю, але і визначених.

Послідовність Ґоломба — це когерентність, де та всі подальші зустрічі з ментальними речами і, що важливо, сума мінімальна [16]. Крім того, мінімальна сума в цьому сценарії нереальна. Наслідки представлені тут у фрагментах, що відповідають акторському сценарію та з використанням шаблонів дозволу.

Перші рішення Голомба були:

$$g3=\{1, 2\}$$

$$g4=\{1, 3, 2\}$$

$$g5=\{1, 3, 5, 2\}$$

$$g6=\{1, 3, 6, 5, 2\}$$

$$g7=\{1, 3, 6, 8, 5, 2\}$$

з відповідними послідовностями суми :

$$G3=\{0, 1, 3\}$$

$$G4=\{0, 1, 4, 6\}$$

$$G5=\{0, 1, 4, 9, 11\}$$

$$G6=\{0, 1, 4, 10, 15, 17\}$$

$$G7=\{0, 1, 4, 10, 18, 23, 25\},$$

за модулем ці довжини дають послідовності

$$\{0, 1, 0\} \pmod 3$$

$$\{0, 1, 0, 2\} \pmod 4$$

$$\{0, 1, 4, 4, 1\} \pmod 5$$

$$\{0, 1, 4, 4, 3, 5\} \pmod 6$$

$$\{0, 1, 4, 3, 4, 2, 4\} \pmod 7.$$

Тому жодна з цих послідовностей Голомба не є нормальною (послідовності підсумовування не мають явного введення). Як зазначалося, для послідовності Голомба не було створено модульного сценарію. Однак ці послідовності можна завершити у звичайних послідовностях, додавши комп'ютери, таким чином збільшуючи довжину та модуль, як показано нижче:

$C3$={0, 1, 3, 2} mod 4

$C4$={0, 1, 4, 6, 2, 3, 5} mod 7

$C5$={0, 1, 4, 3, 5, 2} mod 6

$C6$={0, 1, 4, 10, 3, 5, 2, 6, 7, 8, 9, 11} mod 12

$C7$={0, 1, 4, 10, 2, 7, 9, 3, 5, 6, 8, 11, 12, 13, 14, 15} mod 16.

Згадаймо іншу послідовність, яка має чіткі часткові суми {1, 2, 4, 8, 16, 32,...} − послідовність Log. Ця послідовність дуже вільна. Тому часто хочемо мінімізувати розмір останнього введення. Іншою структурою послідовності з чіткими частковими сумами є економна послідовність [19]. Також для цієї послідовності ми використовуємо процедуру заповнення послідовності іншими комп'ютерами. Починаємо з визначення G_0, (тобто список відновлень для нульового процесу): $g(x)=\min\{j \epsilon N+\}$, таке для всіх a_1, a_2, b_1, b_2 виконуються умови $(1\le a_1\le b_1\le x)$ $(1\le a_2\le b_2\le x)$ $((a_1\ne a_2)$ or $(b_1\ne b_2))$ маємо $\sum_{l=a_1}^{b_1} g(l) \ne \sum_{l=a_2}^{b_2} g(l)$. Якщо $\sum_{l=1}^{x} g(l) < n$, потім $G_0(x)\sum_{l=1}^{x} g(l)$, інакше {$G0(x)=\min\{j \in N_+$-{$G_0((1), \ldots, G_0(x-1))$}}.

Від визначення вище бачимо, що для великих значень n (тобто n>289), перші 16 елементів в G_0 для ощадливої схеми відновлення є: 1, 3, 7, 12, 20, 30, 44, 65, 80, 96, 122, 147, 181, 203, 251, 289.

Потім для n=16, G_0={1, 3, 7, 12, 2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15}. Послідовності [17] модуля подібні до послідовностей Голомба. У цьому типі послідовності сума елементів незначна.

Приклади перших послідовностей модулів:

$m3$={1, 2} mod 3

$m4$={1, 4, 6} mod 7

$m5$={1, 6, 3, 10} mod 11

$m6$={1, 3, 7, 17, 12} mod 18

$m7$={1, 3, 23, 7, 17, 12} mod 24

Висновки до розділу 2

Розрахунок оптимальних рядків стає можливим після подальшого формулювання задачі в ланцюжках і комбінаціях рядків. Щоб вивчити балансування навантаження, розглядаємо комп’ютер, який отримує процеси від кількох комп’ютерів, що перемикаються після відмови.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Навантаження з компенсацією на лінійці Голомба

Як обговорюється, кожна послідовність ($r1$, …, rn – 1) визначає n – 1 ланцюги (лінійки):

C_i={r_1+…+r_i, $r2$+…+r_i,…, r_i} i=1, …, n–1. Давайте розглядати найбільш завантажений комп'ютер. Якщо ми, наприклад, вибирають три ланцюги $Ci1$, $Ci1$ і, $Ci3$ це вимагатиме $|C_{i1} \cup C_{i2} \cup C_{i3}|$, щоб вимкнені процеси $y_{i1,1}$, $y_{i2,1}$, , і $y_{i3,1}$ комп'ютері y, отримали аварією усіх комп'ютерів в списках C_{i1}, C_{i1} і, C_{i3}.

Очевидно, якщо списки не перетинаються, таким чином $|C_{i1} \cup C_{i2} \cup C_{i3}|$, Те, що не можна відокремити, так це те, що існують комп’ютери, вимкнення яких має багаторазовий вплив на переміщення процесів до у, і кількість необхідних вимкнень є більшою чи меншою.

Тож пізніше $|C_{i1} \cup C_{i2} \cup C_{i3}|$ терпить невдачу, ми отримуємо 1+{$i1$, $i2$, $i3$} =4 процеси на y.

Взагалі ми маємо наступне:

Теорема: Нехай I= {$i1$, …, is} (для s > 0), є певною к–тю процесів, якими усі є переміщено в комп'ютер y, якщо усі комп'ютери в $\bigcup_{i \in I} \tilde{N}_i$ вимкнені. Для схеми відновлення R і якщо p вимкнення відбуваються, маємо $L(n, p, R) = 1 + \max_i \left\{ i = |I|; \forall I; \left| \bigcup_{i \in I} C_i \right| \le p \right\}$.

Доказ: Взагалі, множина $\left| \bigcup_{i \in I} C_i \right|$ відповідає випадку $\left| \bigcup_{i \in I} C_i \right|$ = p, комп'ютери вимкнені. Щоб розрахувати найгірший випадок, нам потрібно перевірити всі з’єднання $\left| \bigcup_{i \in I} C_i \right|$ з $\left| \bigcup_{i \in I} C_i \right| \le p$. Тоді у нас є процеси на тому самому комп’ютері. Потім переформулюємо задачу для алгоритму.

Знаходяться в повних можливих об'єднаннях $2^{n-1} - 1$ великих кількостей $C1, \ldots, Cn - 1$, які позначаємо об'єднання близьким, а описуємо, що кожне об'єднання має дві властивості, які представляють інтерес для наших цілей. Можливо, могли б перефразувати визначення з точки зору розміру та кількості великих кількостей:

$$L(n, p, R) = 1 + \max_i \left\{ i = |I|; \forall I; \left| \bigcup_{i \in I} C_i \right| \le p \right\}.$$

Від об'єднань $\left| \bigcup_{i \in I} C_i \right|$ робиться близько підряд $C_1, \ldots, Cn-1$, додаючи попередньо зроблені об'єднання.

Можемо упорядкувати послідовності $(n-1)!$ $(0, R_1, \ldots, R_{n-1})$ разом із усіма можливими коротшими рядками у вбудованому дереві, де два рядки пов’язані один з одним, якщо вони ідентичні, за винятком того, що в одному з них відсутній останній запис.

Отже $(0, R_1, \ldots, R_k)$, і $(0, P_1, \ldots, P_l)$ пов'язані, якщо $R_i = P_i$ хоча $i = 1, \ldots, \min(k, l)$ і $k - l = 1$. Порожня послідовність є коренем дерева, а всі шаблони оновлення є листами. Називемо це дерево деревом послідовності (рис. 3.2).

По суті, нам потрібно перевірити всі можливі зв’язки для кожної послідовності. Однак існує кілька способів усунення великих наборів послідовностей і об'єднань.

Визначення. Говоримо, що *BV* щільна аж до *q*, якщо є схема відновлення *R* таким чином $L(n, i, R) = BV(i)$ *для* $i = 1, \ldots, q$.

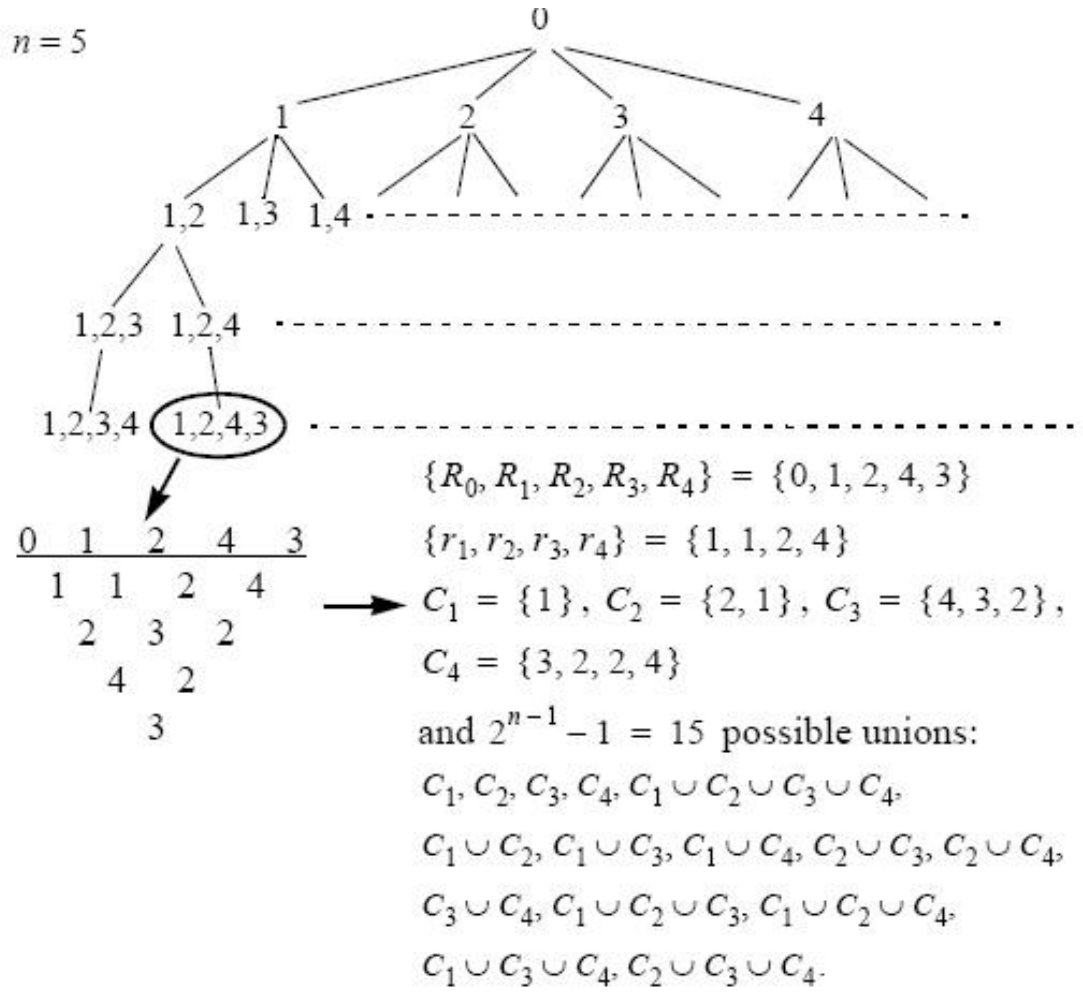


Рис. 3.2. Приклад дерева послідовностей для n=5

Анотація 3.1: Припускають, що *BV* щільна аж до *q*. Потім усі часткові суми *r* аж до потреби *BV (q)* є чітким, якщо *R* − *n*−оптимальна схема відновлення. Іншими словами $\sum_{i=a}^{b} r_i$, номери для усіх а і *b* $1 \le a \le b \le BV(q)$, чіткі.

Доказ: Припускають протиріччя, тобто, що дві часткові суми однакові, скажемо в ряду *i* і *j*. Припускають, що $i \ge j$. Потім ми повинні $i(i+1)/2-1$ того, процеси на комп'ютері з великими навантаженнями будуть відключені вимиканням. Це суперечить оптимальності *R*. Це означає, що ми можемо почати з попередньо розрахованих модульних послідовностей [17]. Говоримо, що такі послідовності задовольняють модульну умову. Якщо модульна послідовність має чіткі послідовності аж до чисел у послідовності,

ці початкові m чисел можна використовувати як початок послідовності та залишатися (n–m–1)! Послідовність для знаходження оптимальності.

Тут досліджуємо для якого значення n і q, зв’язаного MV є щільний. Щільна межа під назвою SV. Напруженість MV, витікає із заздалегідь зробленої послідовності.

Отже, якщо маємо одну схему відновлення, яка йде за MV аж до q, можемо відкинути усі схеми відновлення, де не усі часткові суми аж до q чіткі.

Наступний алгоритм знаходить усі послідовно оптимальні шаблони оновлення. Якщо є попередня послідовність, яка має p-оптимальність, можемо негайно припинити дослідження послідовності, яка не має p-оптимальності.

Також можете перемістити всю гілку дерева послідовностей, тобто гілку з ідентичною основною послідовністю. Це наслідок послідовного балансування навантаження. Якщо він визначає, що послідовність не є оптимальною, пошук переходить до наступної гілки.

У багатьох розподілених системах і кластерах розробник повинен надати схеми відновлення. Такі схеми визначають, як перерозподіляється навантаження, коли один або кілька комп’ютерів виходять з мережі.

Мета полягає в підтримці рівномірного розподілу навантаження між усіма комп'ютерами, навіть коли найбільш несприятливі комбінації комп'ютерів відключені (табл. 3.1). Це особливо важливо в системах реального часу.

Таблиця 3.1

Оптимальні множини

Число вузлів	Максимальне число вимкнених вузлів SV, де співпадає з MV	Приклади оптимальних послідовностей
4	3	0, 1, 3, 2
5	4	0, 1, 3, 2, 4
6	5	0, 1, 3, 5, 4, 2
7	6	0, 1, 4, 6, 2, 3, 5
8	7	0, 1, 3, 7, 5, 6, 2, 4
9	8	0, 1, 3, 7, 5, 2, 8, 6, 4
10	7	0, 1, 5, 3, 2, 9, 7, 8, 4, 6
11	7	0, 1, 6, 3, 10, 2, 4, 5, 7, 8, 9
12	8	0, 1, 4, 9, 11, 6, 3, 2, 5, 7, 8, 10
13	8	0, 2, 7, 6, 3, 10, 5, 1, 4, 8, 9, 11, 12
14	8	0, 6, 1, 5, 3, 11, 2, 4, 7, 8, 9, 10, 11, 13
15	8	0, 1, 6, 8, 4, 11, 2, 3, 5, 7, 9, 10, 12, 13, 14
16	8	0, 1, 6, 8, 4, 11, 2, 3, 7, 5, 9, 10, 12, 13, 14, 15
17	8	0, 1, 6, 8, 4, 13, 2, 3, 7, 5, 9, 10, 11, 12, 14, 15, 16
18	11	0, 2, 11, 8, 3, 7, 5, 1, 4, 6, 9, 10, 12, 13, 14, 15, 16, 17
19	12	0, 1, 8, 12, 6, 3, 11, 2, 4, 5, 7, 9, 10, 13, 14, 15, 16, 17, 18
20	13	0, 1, 8, 13, 4, 10, 7, 6, 2, 3, 5, 12, 11, 9, 16, 14, 15, 17, 18, 19
21	13	0, 1, 9, 11, 7, 14, 3, 2, 5, 6, 8, 10, 12, 13, 15, 16, 17, 18, 19, 20

Розглянемо *n* однакових комп’ютерів, на кожному з яких виконується один процес. Усі процеси виконують однакову кількість роботи. Схеми відновлення, які гарантують оптимальне поширення навантаження в найгіршому випадку, коли *x* комп'ютерів вимкнено, називаються оптимальними схемами відновлення для значень *n* і *x*. Тут представили алгоритм, який створює оптимальні схеми відновлення для будь-якої кількості простоїв. Складність алгоритму - експоненціальна функція.

Наразі оптимальні схеми оновлення розраховано максимум для 26 комп’ютерів. Розраховуються оптимальні послідовності, тобто такі, які забезпечують оптимальне навантаження поширення для будь-якої кількості пошкоджених комп’ютерів. Щоб використовувати ці схеми, повинен існувати певний механізм виявлення несправностей комп'ютера. Це можна зробити різними способами, наприклад, регулярно роблячи перерви на прийом їжі. У цьому випадку зовнішні системи живлення є даними всередині вузла в кластері. Виявленням помилок часто керує програмне забезпечення кластера, і в таких випадках можна безпосередньо застосувати оптимальні схеми оновлення.

Використовуючи статичні діаграми, знаємо, якому комп’ютеру в процесі потрібно буде перезапустити список оновлень, продовжуючи поширювати поточний стан процесу до наступного вузла в ланцюжку, таким чином даючи час для перезавантаження процесу в разі збою. . Це неможливо, коли використовуємо динамічні схеми, і тоді не знаємо, на якому вузлі слід перезапустити процес.

3.2. Неатомні навантаження

Поки що розглядали випадок, коли вся робота, виконана комп’ютером на вузлі, має бути передана як одна атомна одиниця. Звичайно, на кожному комп’ютері може бути більше одного процесу, і в деяких випадках ці процеси можуть перерозподілятися незалежно один від одного, у такому випадку для кожного процесу існує єдиний список оновлень. Розглянемо випадок, коли на кожному комп’ютері є p процесів. Вважаємо навантаження рівномірно розподіленим між процесами. Попередні дослідження показали, що якщо кожен комп’ютер має велику кількість процесів, завдання визначення оптимальних моделей оновлення стає менш важливим [6]. Причина в тому, що інтуїтивні рішення стають кращими. Отже, важливість отримання оптимальних схем відновлення є найбільшою, коли p дорівнює n. Отримали результати для p=1.

Техніка, використана для отримання цих результатів, також корисна, коли враховуємо $p>1$. Тепер визначимо граничні вектори типу p, тобто $p=1$. Пов’язаний вектор p-типу виходить у наступному напрямку (першим входом граничного вектора є вхід 1):

1. $t=p$; $i=1$; $j=1$; $r=1$; $v=1$

2. Якщо $r=1$ потім $t=t+1$; $r=j$; $v=v+1$ інший $r=r-1$

3. Нехай вхід i в пов'язаний вектор типу p має значення t/p

4. $i=i+1$

5. Якщо $v=p$ і $r=1$ потім $j=j+1$; $v=0$

6. Повертаємося до 2 вектору межі, який не має бажаної довжини.

Пов’язаний тип вектора два (завантаження запису на кожну машину, нормалізованого до одиниці, коли всі машини працюють) : {3/2, 4/2, 4/2, 5/2, 5/2, 6/2,6/2, 6/2, 7/2, 7/2, 7/2, 8/2, 8/2, 8/2, 8/2, 9/2, 9/2, 9/2, 9/2, 10/2,…

Пов'язаний вектор типу трьох: {4/3, 5/3, 6/3, 6/3, 7/3, 7/3, 8/3, 8/3, 9/3, 9/3, 9/3, 10/3, 10/3, 10/3, 11/3, 11/3, 11/3, 12/3…

Пов'язаний вектор типу чотирьох: {5/4, 6/4, 7/4, 8/4, 8/4, 9/4, 9/4, 10/4,10/4, 11/4, 11/4, 12/4, 12/4, 12/4, 13/4,…

Для $p\le n$, заздалегідь визначаємо схему відновлення − відновлення $p-$процес схема − оптимальна, поки більшість $\log_2 n/p$ комп'ютерів вимикається [6]. Називаємо цей *p-процесом*, оскільки розглядаємо випадок, коли на кожному комп’ютері є p процесів, коли всі комп’ютери працюють. Тепер визначаємо схему оновлення для процесу p. Схема, визначена when $p\le n$. Ця схема відновлення оптимальна також, коли значно більш ніж $\log_2 n/p$ комп'ютерів вимикаються.

Нехай R_0 є схема оптимального відновлення для деяких n і $R_0(k)$ k є входом в R_0. Нехай потім $R_{i,j}$ означає оптимальний список відновлень для процесу j $(0\le j<p)$ на комп'ютері i $(0\le i<n)$. Визначаємо $R_{i,j}$, який виходить з R_0, тобто схему відновлення для нульового процесу в одному процесі схеми відновлення, зазначеної в табл. 3.2.

Якщо $R_0(k)+j\lfloor n/p\rfloor<n$, потім $Ri, j(k) =(R_0(k)+i+j\lfloor n/p\rfloor)$ mod n, окрім $R_i, j(k) =(R_0(k)+i+j\lfloor n/p\rfloor+n)$ mod n, де $R_i, j(k)$ означає ціле число k в списку для процесу j на комп'ютері i.

Таблиця 3.2

Список відновлення, коли n=11 і p=2.

$R_{0,0}$	1, 6, 3, 10, 2, 4, 5, 7, 8, 9
$R_{0,1}$	6, 0, 8, 4, 7, 9, 10, 1, 2, 3
$R_{1,0}$	2, 7, 4, 0, 3, 5, 6, 8, 9, 10
$R_{1,1}$	7, 1, 9, 5, 8, 10, 0, 2, 3, 4
$R_{2,0}$	3, 8, 5, 1, 4, 6, 7, 9, 10, 0
$R_{2,1}$	8, 2, 10, 6, 9, 0, 1, 3, 4, 5
$R_{3,0}$	4, 9, 6, 2, 5, 7, 8, 10, 0, 1
$R_{3,1}$	9, 3, 0, 7, 10, 1, 2, 4, 5, 6
$R_{4,0}$	5, 10, 7, 3, 6, 8, 9, 0, 1, 2
$R_{4,1}$	10, 4, 1, 8, 0, 2, 3, 5, 6, 7

3.3. Поняття лінійки Голомба

Distributed.net, найвідоміша група розподілених обчислень, 15 лютого 2011 р. офіційно запустив новий проект, метою якого є пошук оптимальної лінійки Голомба з 27 поділками. Проект ще не завершений, хоча планується відкрити новий проект із 28 відділеннями.

Що таке лінійка Голомба? У математиці цим терміном називають послідовність цілих чисел, у яких жодна пара не відрізняється одна від одної на однакове число. Концептуально це відповідає лінійці з маркерами, розташованими таким чином, що відстань між будь-якою парою маркерів завжди унікальна.

Оптимальною лінійкою Голомба є найкоротша лінійка Голомба для заданої кількості поділок. Складність пошуку такої лінії зростає експоненціально із збільшенням кількості поділок, тому Distributed.net взявся за цей проект.

Оскільки лінійки Голомба, окрім використання в схемах реконструкції в розподілених обчисленнях, мають багато застосувань у розташуванні

датчиків у рентгенівській кристалографії та радіоастрономії, а також у комбінаториці, теорії кодування та комунікації, тому цікаво знайти простий але ефективний алгоритм синтезу.

Ще один цікавий додаток, який також має практичне застосування. Це конструкція конференц-залу з рухомими перегородками на пропорційних відстанях, розділених лінійкою Голомба (рис. 3.3).

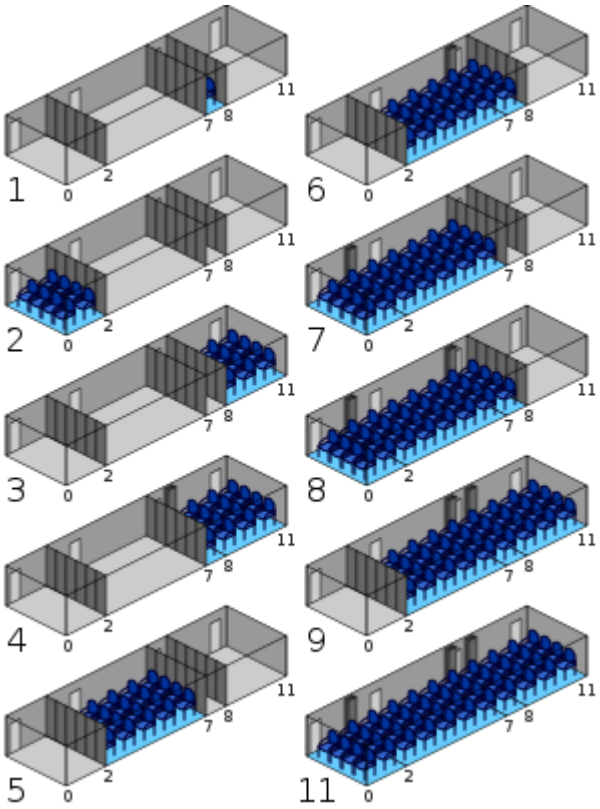


Рис. 3.3. Приклад конференц-залу з поділом на пропорційних відстанях за допомогою лінійки Голомба [0, 2, 7, 8, 11], що дозволяє мати кімнати 10 різних розмірів

3.4. Існуючі алгоритми побудови всієї множини лінійок Голомба

Інша назва лінійок Голомба в науковій літературі - в'язання номерних лінійок (ЧЛВ). Алгоритм вибіркового руху, який використовується в проекті, заснований на «вирощуванні» рядків Голомба натуральних чисел за певними правилами. Його суть полягає в тому, щоб скласти всі суміжні числа, починаючи з одиниці, і знайти серед отриманих сум найменшу суму за

умови, що всі отримані суми в рядку різні. Немає математичних доказів того, що взяття всіх чисел підряд за допомогою алгоритму вибіркового переміщення створює найкоротший ЧЛВ. Тому, на нашу думку, для отримання справді оптимальної лінійки необхідно використовувати модифікований алгоритм вибіркових ходів з пробілами, де номери лінійки та сусідні суми беруться не лише підряд, а й по черзі, два за одним і т.д.

Відомо, що вже при $N > 3$ не можна здійснити упорядкування чисел натурального ряду $1, 2, ..., N$ за операцією додавання так, щоб всі утворені на даній послідовності суми не повторювались. Тому постає питання про впорядкування ланцюжків елементів з мінімальними числовими значеннями так, щоб число таких ланцюжків було мінімальним.

Висновки до розділу 3

Також, враховуючи надто довгий час розрахунку для рядка з 27 поділками - понад рік на тисячах комп'ютерів, виникає необхідність знайти більш ефективний алгоритм з можливістю розпаралелення. Найкращим способом вирішення цієї проблеми є мова логічного програмування Turbo Prolog або Visual Prolog.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Загальна структура написання програми на мові Prolog

Будь-яка мова програмування передбачає наявність моделі машини як пристрою, що використовується для виконання певних операцій над конкретними об'єктами. Чим вищий рівень програмування, тим абстрактнішою стає модель машини і незрозумілішими поняття, з якими працює програміст. Модель логічної задачі має на меті повністю відмовитися від концепції абстрактної віртуальної машини та відмовитися від представлення рішення проблеми у вигляді послідовності конкретних операторів. Це повне абстрагування від машинної моделі та концепції дії на об’єкт призводить до концепції логічного програмування, яке описує дії не як алгоритм розв’язання задач, а як властивість об’єктів і зв’язків між ними. Далі алгоритм вирішення задачі представляється як ланцюжок виведення одних дій-властивостей через інші, від кінцевого результату до вихідних даних.

Щоб написати програму на мові логічного програмування, необхідно спочатку створити систему логічних понять, що виражають суть проблеми. Після встановлення системи абстрагування необхідно описати дії, що виражають властивості введених абстрактних понять, а також необхідні для синтезу алгоритму розв’язання задачі у вигляді ланцюжка логічного висновку. Тому при формулюванні предикатів слід зосереджуватися не на тому, як буде розв’язуватися задача, а на перевірці логічних тверджень, які формулюють зв’язки між поняттями чи подіями.

Пролог надає базові знання, необхідні для написання програм, які розв’язують логічні задачі. Це також дозволяє сформулювати логічні зв’язки між об’єктами в базах даних і ввести логічні поняття про ці об’єкти. Особливої уваги заслуговують бази знань та інтелектуалізовані інформаційні системи, в яких Пролог служить мовою опису знань.

Першою спробою вирішити проблему конкуренції між алгоритмічними мовами програмування була ідея декларативного програмування. Він передбачає, що програміст створює для себе абстрактні структури даних, на яких він вводить основні операції обробки, а також уточнює параметри цих понять. Такі «визначення» буквально переносяться на інші програми і не вимагають додаткового узгодження з іншими поняттями, оскільки вони незалежні та дійсні лише для тих структур даних, для яких вони сформульовані. Проблема залишається відкритою лише тоді, коли різні структури даних взаємодіють.

Очевидно, що описати певні операції над абстрактними структурами даних без засобів логіки та теорії предикатів нереально. Логіка стала фундаментальним елементом декларативного програмування. Створення семантики декларативної мови було нерозривно пов'язане з формулюванням визначень предикатів і частіше називалося логічним програмуванням. Першою декларативною логічною мовою програмування була мова «PROgramming in LOGic» (Prolog). Отже, в декларативному логічному програмуванні поняття підпрограми замінюється поняттям «логічного предиката», який визначає деяке визначення на абстрактній структурі даних, а взаємодія підпрограм замінюється логічними зв’язками між визначеннями предикатів.

Ідея логічного програмування, що передбачає, що змінні в підпрограмі не поділяються на вхід і вихід, заслуговує окремої теми. Вихідною змінною може бути будь-яка невідома змінна. Отже, з правильно заданими визначеннями предикатів невідомі змінні можуть бути логічно виведені з відомих. Найпростіша процедура застосування лежить в основі програм Ppolog.

Значним досягненням логічного програмування є те, що в ньому чітко встановлені зв'язки між логічним змістом підпрограм і їх мовними властивостями. Завдяки такому підходу алгоритм лінгвістично зрощується з

ім'ям-ключем отриманого предиката, завдяки чому в складанні програми бере участь не тільки образне мислення, а й вербальне (словник). Таке поєднання ключових слів мови з логічними поняттями призвело до можливості використання рекурсивних визначень. Рекурсивне визначення – це визначення, яке використовує поняття, які, у свою чергу, рано чи пізно формулюються через поняття, яке зараз визначається. Такі визначення значно спростили логічне програмування, оскільки розв’язування задачі розмірності N зводиться до розв’язування тієї ж задачі менших розмірностей, тому достатньо ввести елементарний (тривіальний) розв’язок задачі та правило отримання іншого розв’язку У програму N. Цей трюк став основним методом програмування в декларативних мовах програмування. У результаті взаємний логічний опис задачі перетворився на програму її вирішення. Все, що потрібно було зробити, це «сформулювати» задачу в термінах введених предикатів.

Програма на мові логічного програмування не визначає алгоритм обробки даних. Оголошує логічні зв’язки між зазначеними подіями, властивостями, об’єктами та діями. Зв'язок представляється як предикат, що містить конкретні логічні поняття: дії, ознаки, об'єкти, дії у вигляді власних змінних. Наприклад, $X=Y+Z$ можна розглядати як предикат, у якому права сторона відома, а ліва невідома. Тоді, щоб предикат був істинним, ліва і права частини рівняння повинні бути рівними. Якщо відомі обидві сторони рівняння, цей предикат можна розглядати як логічну умову, яка може бути істинною. Сам предикат має чітке, логічне визначення в термінах абстрактних змінних, з якими він оперує. У цьому прикладі цей предикат можна назвати «додавання двох змінних».

Логічні умови, накладені на змінні у змісті предиката, визначають його істинність. Програма — це просто набір предикатів, які посилаються один на одного і всі закінчуються крапкою. Кожну з них, залежно від того, які змінні визначені, а які ні, можна інтерпретувати наступним чином:

- Якщо відомі *X, Y,* Z,... то чи виникне подія *f(X,Y,Z,...)*? (або чи збережеться властивість *f(X,Y,Z,...)*?, або чи існуватиме об'єкт *f(X,Y,Z,...)*? і т.д.)
- Якщо відомо, що подія *f(X,Y,Z,...)* відбулася (або є властивість *f(X,Y,Z,...)*, або якщо об'єкт *f(X,Y,Z,...)* існує), то якими повинні бути *X, Y, Z,...* ? Звичайно, розв’язок системи предикатних рівнянь може існувати і вбудований у програму логічний механізм інтерпретації знайде його, але про це під час програмування краще не думати.

Насправді програма складається з багатьох розділів, і кожен розділ містить багато визначень, характерних для цього розділу. Якщо програма не має розділів, окрім *“goal”*, то її виконання нічим не відрізняється від звичайного лінійного алгоритму, наприклад:

goal X=5, Y=6, Z=X+Y, write(“ Z=“,Z).

Отже, в “*goal*” задається мета програми - обчислити деякий предикат. В розділі “*clauses*” визначається логічна модель проблеми, що складається з фактів і правил. Для кожного факту існує логічне тлумачення в концепціях, на яких базується модель проблеми. Правило фактично описує алгоритм розв’язування задачі через перелік інших простіших задач, які необхідно розв’язати.

При написанні програми на декларативній мові, такій як Prolog, є дві можливості. Перший – загальний *забороняється конкретне алгоритмічне мислення,* ускладнює побачити логічні зв'язки і сформулювати визначення у вигляді правил. Коректність програми повинна повністю забезпечуватися правильними визначеннями логічних понять і предикатів, з якими програма співпрацює.

Як видно з програми, розділ *domains* містить визначення типів даних; розділ “*predicates*” - опис форми предикатів (виражають поняття, події, зв’язки тощо) і типів даних, що з’являються в предикатах. Розділ *clauses* представляє зміст програми, оскільки визначає найбільш логічні умови і

зв'язки, які створюють логічну модель завдання. Як видно з цього розділу, кожен логічний вираз закінчується крапкою. Можливі два типи виразів:

1. *P(X1,X2,...,Xn).* - “факти”

2. *P(X1,X2,...,Xn):-Q1,Q2,...,Qm.* - “правила”

Коли предикати цього типу виконуються, щоразу а *покоління змінних,* який зникає, коли програма досягає кінцевої точки предикату. Тому змінні в предикатах відрізняються лише під час виконання даного предиката.

Перший тип предикатів називається фактом. Факт встановлює тісний зв'язок між значеннями його змінних. Факти з однаковими назвами групуються та розглядаються як альтернативи (з умовою “*або*”): якщо є два факти з однаковою назвою і різними значеннями змінних, то комбінації значень при визначенні змінних можуть бути взяті повністю з одного факту або повністю з іншого факту. Вирази другого типу, що містять підрядок “:-” називаються правилами. Позначення “:-” між головою і тілом правила використовується як зв'язка “*якщо*”, “*тоді, коли*”, “*необхідно*” і т.д. Зміст положення полягає в наступному. Щоб предикат *P* був істинним, усі предикати, перераховані в правій частині правила, мають бути істинними.

Як не дивно, сформулювати поняття, що використовуються в алгоритмічних мовах, найскладніше для декларативних мов програмування. Розглянемо предикат створення циклу: *cycle(змінна циклу, початкове значення, кінцеве значення)*, який наведений у програмі. Він складається з двох альтернативних правил: призначити значення змінній циклу або спробувати перейти до нового значення. Шаблон *cycle(N,Y,N)* означає, що якщо перший параметр невідомий, але відомі другий і третій, то першому параметру буде присвоєно значення третього параметра, оскільки вони позначаються однією змінною N.

Процес отримання вихідних даних програми Prolog передбачає виконання певного алгоритму на комп’ютері. Оскільки Пролог є не алгоритмічною, а декларативною мовою програмування, алгоритм її

виконання залежить від алгоритму інтерпретації предикатів, з яких складається програма.

Методологія логічного програмування - це підхід, згідно з яким програма містить опис проблеми в термінах фактів і логічних формул, а система вирішує проблему за допомогою механізму логічних міркувань.

Ідея використання мови логіки предикатів першого порядку як мови програмування народилася в 1960-х роках, коли були створені численні автоматичні системи доведення теорем і відповідні системи на їх основі. Суть цієї ідеї полягає в тому, що програміст не показує машині послідовність кроків, що ведуть до вирішення проблеми, як це має місце у всіх процедурних мовах програмування, а описує логічною мовою властивості області інтересу, іншими словами, він описує зміст проблеми. Решта властивостей машина знайшла б сама, побудувавши логічні висновки.

Після відкриття методу розв’язання, як запропонував Корделл Грін, він незабаром був використаний як основа для нової мови програмування. У 1971 році Ален Кольмерауер створив логічну мову програмування Prolog. Логічне програмування досягло піку популярності в середині 1980-х років, коли воно послужило основою для розробки програмного та апаратного забезпечення для обчислювальних систем п'ятого покоління. Сьогодні починається друге життя цієї унікальної мови.

Для методології логічного програмування властиві два основні методи:

- метод однорідного застосування механізму логічного доказу до всієї програми;
- метод уніфікації – механізм зіставлення шаблонів, який використовується для створення та декомпозиції структур даних.

Опишемо більш систематично процес програмування на мові Пролог.

Рішення завдань. Кінцевою метою написання комп'ютерної програми є створення інструменту, призначеного для вирішення проблеми в певній області застосування. Область застосування - це абстрактна частина або

область знань. Структура домену програми складається з сутностей, функцій і зв’язків, які мають відношення до цього домену. В успішній комп’ютерній програмі структура прикладної області моделюється таким чином, що поведінка програми відтворює деякі важливі аспекти цієї структури. Наприклад, зв’язок, який існує між деякими сутностями домену програми, повинен, за аналогією, зберігатися для нотацій, що представляють ці сутності в програмі, що моделює цю область.

Програмування. Процес написання програми на Пролозі принципово схожий на процес побудови теорії логіки предикатів. Програміст аналізує важливі сутності, функції та залежності в області застосування та вибирає для них позначення. Програміст визначає кожну функцію та кожне співвідношення. Для відношень було вказано, які конкретні реалізації дають істину, а які дають хибність. Програміст аксіоматично визначає кожне співвідношення за допомогою правил і фактів Прологу. Аксіоматичне визначення відношення матиме успіх, якщо воно відтворює зміст семантичного визначення. Набір аксіоматичних визначень усіх зв'язків, що мають відношення до даної тематичної області, становить програму, що моделює структуру цієї області.

Для запиту набору правил і фактів програміст або користувач використовує інтерпретатор Прологу. Набір, що складається із запиту, набору програмних правил і фактів, а також мовного інтерпретатора, можна розглядати як алгоритм розв’язання задач у прикладній області. У цьому випадку запити, правила і факти програми являють собою вихідні формули алгоритму, а інтерпретатор містить правила перетворення цих формул. Перекладач діє як активна сила, яка робить висновки з програмних принципів і таким чином реалізує конкретні програмні принципи.

Р. Ковельський описує сутність логічного програмування за допомогою виразу:

Алгоритм=Логіка+Керування.

Подібно до того, як Lisp приховав пристрій пам’яті комп’ютера від програміста, Prolog дозволив програмісту не турбуватися (без потреби) про потік керування програмою.

4.2. Логічна структура програми знаходження лінійок Голомба

Логічна програма складається з набору інструкцій, і обчислення, що виконуються під керуванням такої програми, є логічним завершенням деякої цільової інструкції з використанням лінії Голомба мінімальної довжини.

У нашому випадку це система нерівностей для N елементів прямої Голомба.

Логічний висновок здійснюється згідно з визначенням ліній Голомба та згідно з принципами математичної логіки, і ці правила застосовуються автоматично, що є однією з найважливіших переваг цього алгоритму.

Наприклад, напишемо систему нерівностей для простої лінійки Голомба з кількістю елементів $N = 5$:

X1≠X2;

X1≠X3;

X1≠X4;

X1≠X5;

X2≠X3;

X2≠X4;

X2≠X5;

X3≠X4;

X3≠X5;

X1+X2≠X3;

X1+X2≠X4;

X1+X2≠X5;

X2+X3≠X1;

X2+X3≠X4;

X2+X3≠X5;

X3+X4≠X1;

X3+X4≠X2;

X3+X4≠X5;

X1+X2+X3≠X4;

X1+X2+X3≠X5;

X2+X3+X4≠X1;

X2+X3+X4≠X5;

X1+X2≠X3+X4;

X1+X2≠X4+X5;

X2+X3≠X4+X5;

X2+X3≠X1+X5;

X3+X4≠X1+X5.

Відповідно, за аналогією можна записати систему логічних нерівностей для будь-якої кількості елементів прямої Голомба.

Однак, враховуючи, що зі збільшенням порядку рядка Голомба загальна кількість логічних рівнянь зростатиме на порядок рядка Голомба, єдиним рішенням є створення програми на іншій мові, наприклад C++, яка генеруватиме вміст програми для мови Пролог.

Результати роботи програм представлені в табл. 4.1, де для зручності побудови схем оновлення лінійка Голомба записана у вигляді відстаней. Наприклад, лінійка Голомба з кількістю елементів $N=4$ (1, 3, 5, 2) буде записана у вигляді п’яти відстаней (0, 1, 4, 9, 11), де кожна відстань, це сума попередніх елементів лінійки Голомба.

Таблиця 4.1

Відстані лінійки Голомба для схем відновлення

Кількість відстаней лінійки Голомба	Довжина лінійки Голомба	Відстані лінійки Голомба
1	**2**	**3**
2	1	0 1
3	3	0 1 3
4	6	0 1 4 6
5	11	0 1 4 9 11 0 2 7 8 11
6	17	0 1 4 10 12 17 0 1 4 10 15 17 0 1 8 11 13 17 0 1 8 12 14 17
7	25	0 1 4 10 18 23 25 0 1 7 11 20 23 25 0 1 11 16 19 23 25 0 2 3 10 16 21 25 0 2 7 13 21 22 25
8	34	0 1 4 9 15 22 32 34
9	44	0 1 5 12 25 27 35 41 44
10	55	0 1 6 10 23 26 34 41 53 55
11	72	0 1 4 13 28 33 47 54 64 70 72 0 1 9 19 24 31 52 56 58 69 72
12	85	0 2 6 24 29 40 43 55 68 75 76 85
13	106	0 2 5 25 37 43 59 70 85 89 98 99 106
14	127	0 4 6 20 35 52 59 77 78 86 89 99 122 127

Продовження табл. 4.1

1¤	2¤	3¤
15¤	151¤	0·4·20·30·57·59·62·76·100·111·123·136·144·145·151¤
16¤	177¤	0·1·4·11·26·32·56·68·76·115·117·134·150·163·168·177¤
17¤	199¤	0·5·7·17·52·56·67·80·81·100·122·138·159·165·168·191·199¤
18¤	216¤	0·2·10·22·53·56·82·83·89·98·130·148·153·167·188·192·205·216¤
19¤	246¤	0·1·6·25·32·72·100·108·120·130·153·169·187·190·204·231·233·242·246¤
20¤	283¤	0·1·8·11·68·77·94·116·121·156·158·179·194·208·212·228·240·253·259·283¤
21¤	333¤	0·2·24·56·77·82·83·95·129·144·179·186·195·255·265·285·293·296·310·329·333¤
22¤	356¤	0·1·9·14·43·70·106·122·124·128·159·179·204·223·253·263·270·291·330·341·353·356¤
23¤	372¤	0·3·7·17·61·66·91·99·114·159·171·199·200·226·235·246·277·316·329·348·350·366·372¤
24¤	425¤	0·9·33·37·38·97·122·129·140·142·152·191·205·208·252·278·286·326·332·353·368·384·403·425¤
25¤	480¤	0·12·29·39·72·91·146·157·160·161·166·191·207·214·258·290·316·354·372·394·396·431·459·467·480¤
26¤	492¤	0·1·33·83·104·110·124·163·185·200·203·249·251·258·314·318·343·356·386·430·440·456·464·475·487·492¤

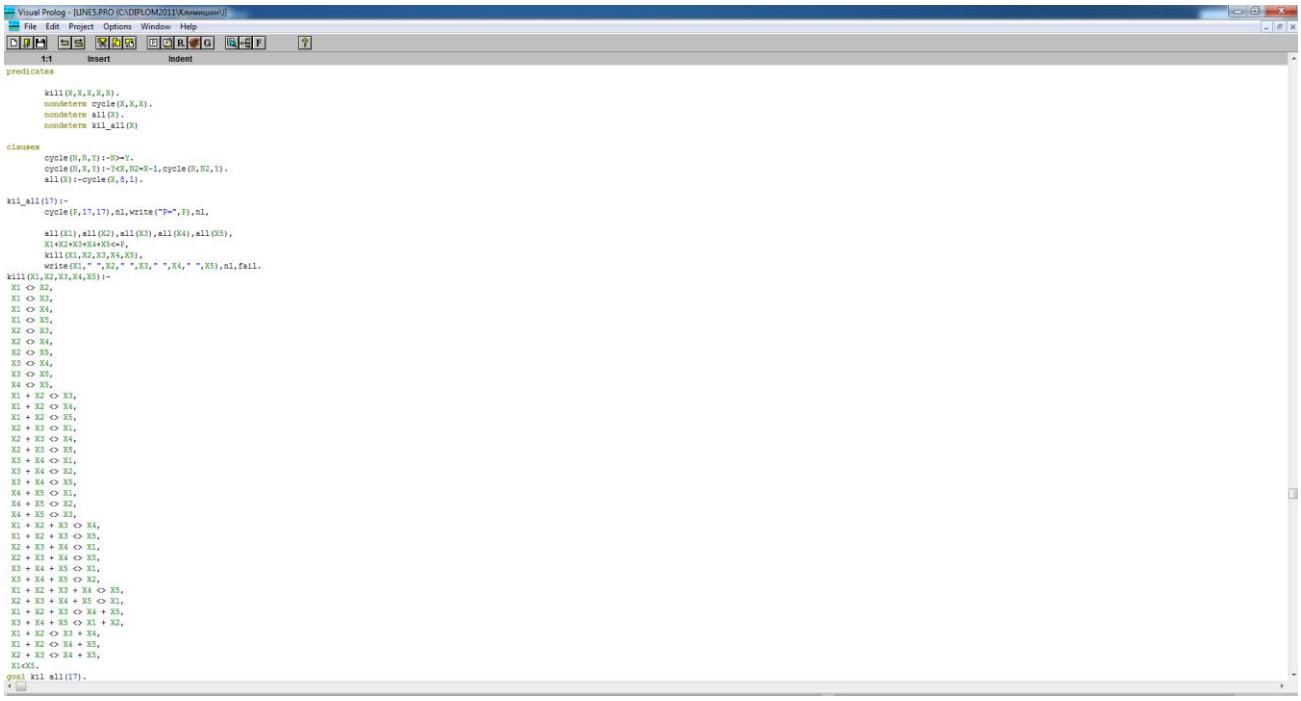
4.3. Опис програмної реалізації та інструкція для користувача

З появою операційної системи Windows 10 принципово змінилися основні принципи створення програм, які тепер можуть мати відмінний і сучасний графічний інтерфейс, можливість підключення і використання звичайних функцій Windows, підтримку роботи в локальних мережах і обміну даними. з іншими програмами під час виконання.

Пряме програмування в Windows можливе за допомогою програмних пакетів, які його підтримують, наприклад Borland Pascal для Windows, Delphi, Borland C++, Visual Prolog. Однак у цьому випадку нас цікавила продуктивність скомпільованого програмного модуля, і в результаті тестування виявилося, що для нашого випадку, де весь алгоритм зводився до простих логічних обчислень, Turbo C++, який будує базу даних логічних інструкції для Visual Prolog 5.1, є найбільш підходящим. Тому було написано дві версії програмного забезпечення: на Turbo C++ і Visual Prolog 5.1, в яких моделі системи відновлення синтезовані на основі ліній Голомба.

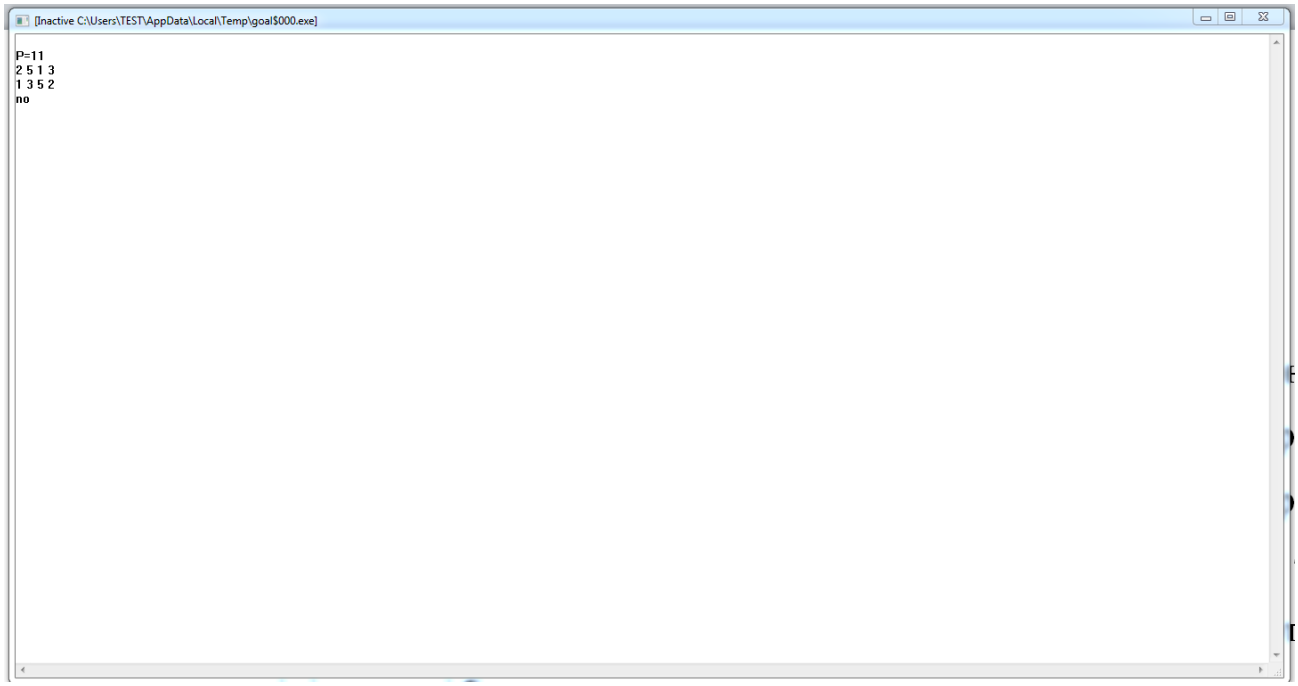
Програма написана на Turbo C++ і Visual Prolog 5.1. Ніяких додаткових компонентів не використовувалося, тільки стандартні. Програми працюють на таких операційних системах: DOS, Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, Windows XP, Windows 7.

Після запуску програми перед початком синтезу лінії Голомба для оновлення діаграм необхідно встановити кількість елементів лінії Голомба в програмі на C++ (додаток А). Результатом є вміст програми в Prolog (Додаток В), який зберігається у файлі з розширенням pro. Потім запустіть Visual Prolog із цього файлу (рис. 4.1).



```
predicates

        kill(X,X,X,X,X).
        nondeterm cycle(X,X,X).
        nondeterm all(X).
        nondeterm kil_all(X)

clauses
        cycle(N,N,Y):-N>Y.
        cycle(N,X,Y):-X<N,N2=X-1,cycle(N,N2,Y).
        all(N):-cycle(N,5,1).

kil_all(17):-
        cycle(F,17,17),nl,write("P=",F),nl,

        all(X1),all(X2),all(X3),all(X4),all(X5),
        X1+X2+X3+X4+X5<=F,
        kill(X1,X2,X3,X4,X5),
        write(X1," ",X2," ",X3," ",X4," ",X5),nl,fail.
kill(X1,X2,X3,X4,X5):-
 X1 <> X2,
 X1 <> X3,
 X1 <> X4,
 X1 <> X5,
 X2 <> X3,
 X2 <> X4,
 X2 <> X5,
 X3 <> X4,
 X3 <> X5,
 X4 <> X5,
 X1 + X2 <> X3,
 X1 + X2 <> X4,
 X1 + X2 <> X5,
 X2 + X3 <> X1,
 X2 + X3 <> X4,
 X2 + X3 <> X5,
 X3 + X4 <> X1,
 X3 + X4 <> X2,
 X3 + X4 <> X5,
 X4 + X5 <> X1,
 X4 + X5 <> X2,
 X4 + X5 <> X3,
 X1 + X2 + X3 <> X4,
 X1 + X2 + X3 <> X5,
 X2 + X3 + X4 <> X1,
 X2 + X3 + X4 <> X5,
 X3 + X4 + X5 <> X1,
 X3 + X4 + X5 <> X2,
 X1 + X2 + X3 + X4 <> X5,
 X2 + X3 + X4 + X5 <> X1,
 X1 + X2 + X3 <> X4 + X5,
 X3 + X4 + X5 <> X1 + X2,
 X1 + X2 <> X3 + X4,
 X1 + X2 <> X4 + X5,
 X2 + X3 <> X4 + X5,
 X1<X5.
goal kil all(17).
```

Рис. 4.1. Програма на Visual Prolog для схем відновлення на основі лінійки Голомба 5-го порядку



```
[Inactive C:\Users\TEST\AppData\Local\Temp\goal$000.exe]

P=11
2 5 1 3
1 3 5 2
no
```

Рис. 4.2. Результат роботи програми на Visual Prolog для схем відновлення на основі лінійки Голомба 4-го порядку

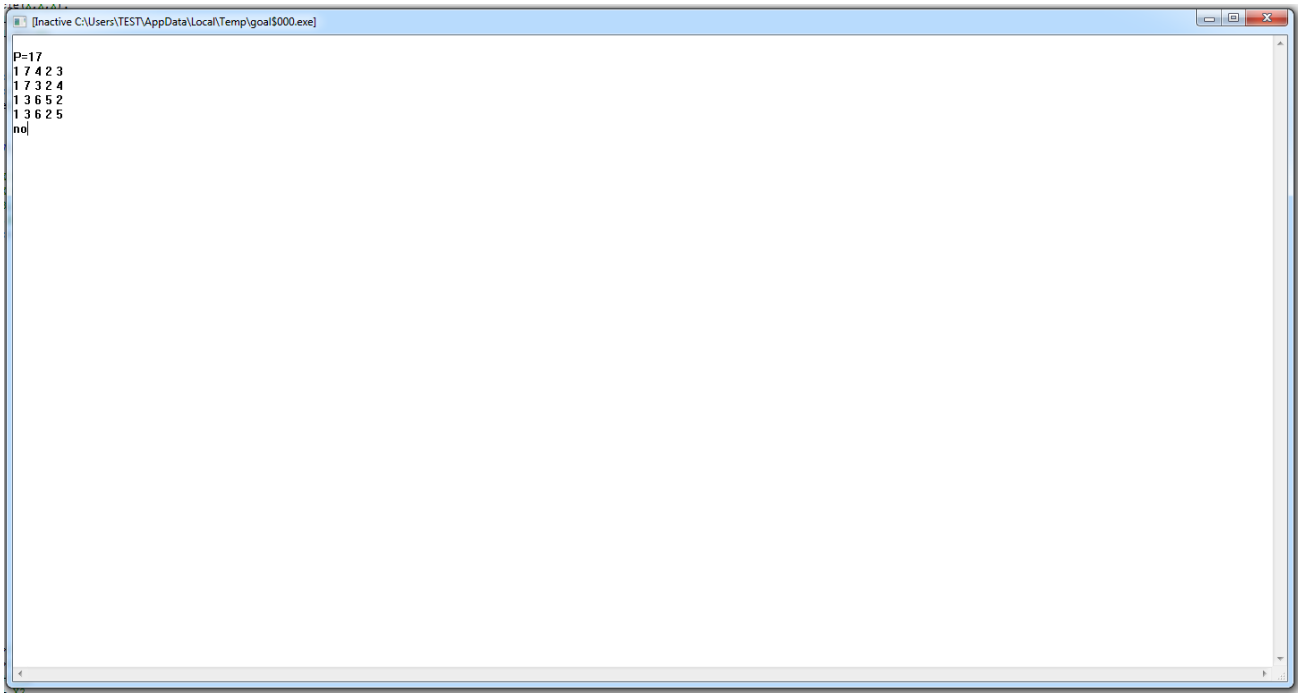


Рис. 4.3. Результат роботи програми на Visual Prolog для схем відновлення на основі лінійки Голомба 5-го порядку

Висновки до розділу 4

Враховуючи, що зі збільшенням порядку лінійки Голомба загальна кількість логічних рівнянь буде зростатиме відповідно до порядка лінійки Голомба, єдиним рішенням є створення програми на іншій мові, наприклад C++, яка генеруватиме вміст програми для мови Пролог.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Опис ідеї проєкту

Продуктивність комп'ютера безпосередньо залежить від тривалості основної операції та кількості основних операцій, які можна виконувати одночасно. Час виконання елементарної операції обмежений часом виконання елементарної операції внутрішнього процесора (тактами процесора). Уповільнення обмежене фізичними міркуваннями, такими як швидкість світла. Щоб оминути ці обмеження, виробники процесорів намагаються реалізувати паралельну роботу всередині чіпа – під час виконання елементарних та базових операцій. Однак теоретично було показано, що стратегія дуже великомасштабної інтеграції є дорогою, а час обчислень залежить від розміру чіпа. Крім цього для підвищення продуктивності комп'ютера використовуються інші методи: конвеєрна обробка (різні етапи окремих команд виконуються одночасно), багатофункціональні модулі (окремі помножувачі, суматори тощо, керовані одним потоком команд).

5.2. Розроблення ринкової стратегії

Все частіше комп'ютер містить «обчислювальні блоки» та відповідну логіку для їхнього підключення. Кожен такий обчислювальний блок має свій процесор і пам'ять. Успіх технології великомасштабної інтеграції у скороченні кількості комп'ютерних компонентів полегшує створення таких комп'ютерів. Крім того, оскільки вартість комп'ютера хоча і приблизно, але пропорційна кількості компонентів, що входять до нього, збільшення інтеграції компонентів дозволяє збільшити кількість процесорів в комп'ютері без істотного збільшення вартості.

Ще однією важливою тенденцією розвитку інформаційних технологій є колосальне зростання ефективності комп'ютерних мереж. Донедавна мережі мали швидкість 1,5 Мбіт/с, а сьогодні існують мережі зі швидкістю кілька

Гбіт на секунду. У міру збільшення швидкості мережі збільшується надійність передачі. Це дозволяє створювати додатки, що використовують фізично розподілені ресурси, якби вони були частиною одного багатопроцесорного комп'ютера. Наприклад, колективне використання ізольованих баз даних, обробка графічних даних на одному або кількох графічних комп'ютерах, а також друк та керування в реальному часі на робочих станціях.

Але є одна серйозна проблема – це перерозподіл навантаження на комп'ютери у разі виходу з ладу одного із процесорів чи всього комп'ютера. Далі виникає проблема розподілу цього навантаження за схемами відновлення.

Метою проєкту є розробка програмного забезпечення для схем відновлення у розподіленій обробці, а саме розробка простого та ефективного алгоритму генерації простих числових переплетень (ліній Голомба) для подальшої генерації оптимальних схем відновлення за критерієм кількості комп'ютерів. підтримується ефективність програмного забезпечення та досліджень, а також необхідність використання створених схем відновлення для кластерних обчислень.

5.3. Розроблення маркетингової програми

Розроблене програмне забезпечення може бути використане для:

– балансування навантаження у кластерних системах:

– онлайн робота з розподілених схем відновлення даних.

Споживачами цього програмного забезпечення можуть бути великі виробничі компанії та дослідницькі центри.

Конкурентом цього продукту є SunSystems, що випускає аналог цієї програми SunCluster.

Основними недоліками аналога є:

– відсутність відкритого вихідного коду програмного забезпечення;

– висока ціна ліцензійного програмного забезпечення.

Основними перевагами аналогового рішення у порівнянні з проектним рішенням є:

– висока швидкість;

– програмний супровід та підтримка.

5.4. Вимоги до технічного та програмного забезпечення

Кластерні системи повинні мати схеми відновлення. Такі схеми визначають, як перерозподілятиметься навантаження у разі виходу з ладу одного або кількох комп'ютерів.

Необхідно завжди розподіляти навантаження рівномірно, навіть за найнесприятливіших поєднань комп'ютерних збоїв.

У даному проєкті представлена модель схеми відновлення даних на основі лінії Голомба, для якої оптимально більша кількість пошкоджених комп'ютерів, порівняно з іншими схемами відновлення.

Висновки до розділу 5

Програмне забезпечення було розроблено та протестовано для демонстрації роботи моделі схеми відновлення на основі лінії Голомба.

Проєктне рішення є конкурентоспроможним та його реалізація цілеспрямованою.

ВИСНОВКИ

Досліджені схеми відновлення на основі лінійок Голомба, і в результаті визначено обмеження їх використання до 492 комп’ютерів у розподіленій системі, оскільки оптимальні лінійки понад 26 поділок невідомі.

Розроблена логічна структуруапрограми Prolog для побудови лінійок Голомба.

Було розроблено та налагоджено програму C++ для побудови тіла логічної програми в Prolog, яка використовується для побудови моделей відновлення розподілених обчислювальних систем на основі лінійок Голомба.

Отримано результати побудови ліній Голомба в програмі Prolog до порядку 24. Для отримання інструкцій, більших за 26, потрібно виконувати програму кілька днів, що вдвічі менше, ніж раніше.

СПИСОК ЛІТЕРАТУРИ

1. Різник В.В. Синтез оптимальних комбінаторних систем. Львів. Вища школа., 1989.
2. Різник О.Я., Бабчанік І., Буцик В.В., Кміть І.А., Троян О.А. Паралельні розподілені обчислення на лінійках Голомба. Збірка наукових праць міжнародної наукової конференції ISDMCI’2011, м.Євпаторія, т.1, с.372-376.
3. Різник О.Я., Бохан М.П., Бурда Ю.І., Пархоменко Н.Є., Яновська Ю.Є. Лінійка Голомба як комбінаторна модель на послідовностях чисел. Збірка наукових праць міжнародної наукової конференції ISDMCI’2011, м.Євпаторія, т.2, с.281-283.
4. Різник О.Я., Гринів Х.М., Дикий А.Я., Козовий М.Ю., Мельник О.М.. Деякі характеристики лінійок Голомба. Збірка наукових праць міжнародної наукової конференції ISDMCI’2011, м.Євпаторія, т.2, с.283-286.
5. Різник О.Я., Скрибайло-Леськів Д.Ю., Перепечай О.С., Тершівський І.Б. Синтез псевдовипадкових послідовностей великої довжини. Технічні вісті. Наково-публіцистичний часопис. 2010/1(31), 2(32), м.Львів, с.109-110.
6. Bertsekas, D.P., Ozveren, C., Stamoulis, G.D., Tsitsiklis, J.N., Optimal Communication Algorithms for Hypercubes, Journal of Parallel and Distributed Computing, 11, 1991, pp. 263−275
7. Chinchani, R., Upadhyaya, S., Kwiat, K., A Tamper−Resistant Framework for Unambiguous Detection of Attacks in User Space Using Process Monitors, in Proceedings of First IEEE International Workshop on Information Assurance IWIA'03, March 24−24, 2013, Darmstadt, Germany, pp. 25−36

8. Cluster and Distributed Computing, Journal of Parallel and Distributed Computing 61(11), 2021, pp. 1680−1691
9. Dimitromanolakis, A, Analysis of the Golomb Ruler and the Sidon Set Problems, and Determination of large, near−optimal Golomb Rulers, Dept. of Electronic and Computer Engineering Technical University of Crete, June, 2022
10. Gelenbe, E., Derochete, D., Performance of Rollback Recovery Systems under Intermittent Failures, Communication of the ACM, 21(6), 2018, pp. 493−499
11. Golomb, S. W., Taylor, H., Cyclic Projective Planes, Perfect Circular Rulers, and Good Spanning Rulers, Sequences And Their Applications – SETA'01, Proceedings of Seta01, Bergen, Norway, May 2001, pp. 166−180
12. Kameda, H., Fathy, E.−Z.S., Ryu, I., Li, J., A performance Comparison of Dynamic vs. Static Load Balancing Policies in a Mainframe − Personal Computer Network Model, Information: an International Journal, 5(4), 2022, pp. 431−446
13. Klonowska, K., Lennerstad, H., Lundberg, L., Svahnberg, C., Optimal Recovery Schemes in Fault Tolerant Distributed Computing, Acta Informatica, 41(6), 2005
14. Klonowska, K., Lundberg, L., and Lennerstad, H., Using Golomb Rulers for Optimal Recovery Schemes in Fault Tolerant Distributed Computing, in Proceedings of the 17th International Parallel & Distributed Processing Symposium IPDPS 2003, Nice, France, April 2003
15. Klonowska, K., Lundberg, L., Lennerstad, H., Svahnberg, C.: Using Modulo Rulers for Optimal Recovery Schemes in Distributed Computing, in Proceedings of 10th International Symposium PRDC 2004, Papeete, Tahiti, French Polynesia, March 2004, pp. 133−142
16. Klonowska, K., Lundberg, L., Lennerstad, H.: Using Golomb Rulers for Optimal Recovery Schemes in Fault Tolerant Distributed Computing, in

Proceedings of 17th International Parallel & Distributed Processing Symposium IPDPS 2003, Nice, France, April 2003, pp. 213, CD−ROM

17. Krishna, C.M., Shin, K.G., Real−Time Systems, McGraw−Hill International Editions, Computer Science Series, 1997, ISBN 0−07−114243−6
18. Lundberg, L., and Svahnberg, C., Optimal Recovery Schemes for High−Availability Cluster and Distributed Computing, Journal of Parallel and Distributed Computing 61(11), 2001, pp. 1680−1691
19. Lundberg, L., and Svahnberg, C., Optimal Recovery Schemes for High−Availability Cluster and Distributed Computing, Journal of Parallel and Distributed Computing 61(11), 2001, pp. 1680−1691
20. Lundberg, L., Svahnberg, C., Optimal Recovery Schemes for High−Availability Cluster and Distributed Computing, Journal of Parallel and Distributed Computing, 61(11), 2001, pp. 1680−1691
21. Microsoft Corporation, Server Clusters: Architecture Overview for Windows Server 2003, Microsoft Corporation, March 2003
22. Reinhardt, S.K., Mukherjee, S.S., Transient Fault Detection via Simultaneous Multithreading, in Proceedings of 27th Annual International Symposium on Computer Architecture (ISCA), Vancouver, British Columbia, Canada, June, 2000
23. Soliday, S. W., Homaifar, A., Lebby, G. L., Genetic Algorithm Approach to the Search for Golomb Rulers,in Proceedings of the International Conference on Genetic Algorithms, Pittsburg, PA, USA, pp. 528−535, 1995
24. Stalling, W., Computer Organization & Architecture, Designing for Performance, Sixth Edition, Prentice Hall, 2023, ISBN 0−13−049307−4
25. Sun Cluster 2.1 System Administration Guide, Sun Microsystems, 1998.
26. Sun Microsystems. Sun Cluster 3.0 Data Services Installation and Configuration Guide, Sun Microsystems, 2020
27. TruCluster, Systems Administration Guide, Digital Equipment Corporation, http://www.unix.digital.com/faqs/publications/cluster_doc

28. http://www.cs.cornell.edu/rdc/mscs/nt98
29. http://www.cuug.ab.ca:8001/~millerl/g3−records.html
30. http://www.distributed.net/ogr/index.html
31. http://www.research.ibm.com/people/s/shearer/grtab.html

ДОДАТОК А. ПРОГРАМА ПОБУДОВИ ТІЛА ЛОГІЧНОЇ ПРОГРАМИ ДЛЯ МОВИ PROLOG ДЛЯ МОДЕЛЕЙ ВІДНОВЛЮЮЧИХ СХЕМ НА ОСНОВІ ЛІНІЙОК ГОЛОМБА НА МОВІ С++

```
# define MAX 99
# include <stdio.h>
# include <conio.h>
# include <string.h>
int N,m,k,l,t,Nr,Nk;
char fname[]="line_enk.txt";
FILE *Fl;
void vvidN(void)
{
   clrscr();
   puts("Програма Климишин Ю.М., 2011 \n");
   puts("N>1=");
   scanf("%d",&N);
}
void kil(void)
{
  int k;
  printf("\nKIL%d(",++Nk);
  fprintf(Fl,"\nKIL%d(",Nk);
  for (k=1;k<N;k++)
    { printf("X%d,",k);
     fprintf(Fl,"X%d,",k);
    }
     printf("X%d):-",k);
     fprintf(Fl,"X%d):-",k);
}
void main(void)
{
     register int i,j;
     vvidN();
     Fl=fopen(fname,"w");
     Nr=0;
     Nk=0;
//     printf("N = %d\n",N);
//     fprintf(Fl," N = %d  \n",N);
     kil();
//   в обох сумах по одному значенню
     for (i = 1;i<N;i++)
                for (j = i+1;j<=N;j++)
                          { printf(   "\n X%d <> X%d ",i,j);
                           fprintf(Fl,"\n X%d <> X%d ",i,j);
                           if(++Nr>=MAX){kil();Nr=0;};
                          }
//   перша сума має більшу кількість співпадань
//   m - число елементів першої суми
//   i - номер першого елемента першої суми
//   l - число елементів другої суми
//   k - номер першого елемента другої суми
     if (N>2)
              { for (l=1; l<=(int)(N/2); l++)
           for (m = l+1; m<N-l+1; m++)
              for (i = 1; i<=N-m+1; i++)
                           for (k = 1; k<=N-l+1; k++)
                           {if (k<i-l+1 || k>=i+m)
                                    {   printf(   "\n X%d",i);
                                      fprintf(Fl,"\n X%d",i);
                                      for (t = i+1; t<i+m; t++)
                                      { printf(   " + X%d",t);
                                       fprintf(Fl," + X%d",t);
                                      }
                                       printf(   " <> X%d",k);
                                       fprintf(Fl," <> X%d",k);
                                       for (t = k+1; t<k+l; t++)
```

```
                                        { printf(   " + X%d",t);
                                         fprintf(Fl," + X%d",t);
                                        }
                                         printf(   ",");
                                         fprintf(Fl,",");
                                         if(++Nr>=MAX){kil();Nr=0;};
                                       }
                                }
                }
//    обидві суми мають однакову кількість елементів
//    l - число елементів
//    i - номер першого елементу першої суми
//    k - номер першого елементу другої суми
      if (N>3)
                { for (l=2; l<=(int)(N/2); l++)
                        {for (i = 1; i<=N-2*l+1 ; i++)
                             for (k = i+l; k<=N-l+1; k++)
                                   {   printf(   "\n X%d",i);
                                     fprintf(Fl,"\n X%d",i);
                                     for (t = i+1; t<i+l; t++)
                                     { printf(   " + X%d",t);
                                     fprintf(Fl," + X%d",t);
                                     }
                                      printf(   " <> X%d",k);
                                     fprintf(Fl," <> X%d",k);
                                     for (t = k+1; t<k+l; t++)
                                     { printf(   " + X%d",t);
                                     fprintf(Fl," + X%d",t);
                                     }
                                      printf(   ",");
                                     fprintf(Fl,",");
                                     if(++Nr>=MAX){kil();Nr=0;};
                                   }
                        }
                }
     fclose(Fl);
     getch();
}
```

ДОДАТОК Б. РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ ПОБУДОВИ ТІЛА ЛОГІЧНОЇ ПРОГРАМИ ДЛЯ МОВИ PROLOG ДЛЯ МОДЕЛЕЙ ВІДНОВЛЮЮЧИХ СХЕМ НА ОСНОВІ ЛІНІЙОК ГОЛОМБА З КІЛЬКІСТЮ ЕЛЕМЕНТІВ $N = 10$

```
KIL1(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10):-
 X1 <> X2,
 X1 <> X3,
 X1 <> X4,
 X1 <> X5,
 X1 <> X6,
 X1 <> X7,
 X1 <> X8,
 X1 <> X9,
 X1 <> X10,
 X2 <> X3,
 X2 <> X4,
 X2 <> X5,
 X2 <> X6,
 X2 <> X7,
 X2 <> X8,
 X2 <> X9,
 X2 <> X10,
 X3 <> X4,
 X3 <> X5,
 X3 <> X6,
 X3 <> X7,
 X3 <> X8,
 X3 <> X9,
 X3 <> X10,
 X4 <> X5,
 X4 <> X6,
 X4 <> X7,
 X4 <> X8,
 X4 <> X9,
 X4 <> X10,
 X5 <> X6,
 X5 <> X7,
 X5 <> X8,
 X5 <> X9,
 X5 <> X10,
 X6 <> X7,
 X6 <> X8,
 X6 <> X9,
 X6 <> X10,
 X7 <> X8,
 X7 <> X9,
 X7 <> X10,
 X8 <> X9,
 X8 <> X10,
 X9 <> X10,
 X1 + X2 <> X3,
 X1 + X2 <> X4,
 X1 + X2 <> X5,
 X1 + X2 <> X6,
 X1 + X2 <> X7,
 X1 + X2 <> X8,
 X1 + X2 <> X9,
 X1 + X2 <> X10,
 X2 + X3 <> X1,
 X2 + X3 <> X4,
 X2 + X3 <> X5,
 X2 + X3 <> X6,
 X2 + X3 <> X7,
 X2 + X3 <> X8,
 X2 + X3 <> X9,
```

```
 X2 + X3 <> X10,
 X3 + X4 <> X1,
 X3 + X4 <> X2,
 X3 + X4 <> X5,
 X3 + X4 <> X6,
 X3 + X4 <> X7,
 X3 + X4 <> X8,
 X3 + X4 <> X9,
 X3 + X4 <> X10,
 X4 + X5 <> X1,
 X4 + X5 <> X2,
 X4 + X5 <> X3,
 X4 + X5 <> X6,
 X4 + X5 <> X7,
 X4 + X5 <> X8,
 X4 + X5 <> X9,
 X4 + X5 <> X10,
 X5 + X6 <> X1,
 X5 + X6 <> X2,
 X5 + X6 <> X3,
 X5 + X6 <> X4,
 X5 + X6 <> X7,
 X5 + X6 <> X8,
 X5 + X6 <> X9,
 X5 + X6 <> X10,
 X6 + X7 <> X1,
 X6 + X7 <> X2,
 X6 + X7 <> X3,
 X6 + X7 <> X4,
 X6 + X7 <> X5,
 X6 + X7 <> X8,
 X6 + X7 <> X9,
 X6 + X7 <> X10,
 X7 + X8 <> X1,
 X7 + X8 <> X2,
 X7 + X8 <> X3,
 X7 + X8 <> X4,
 X7 + X8 <> X5,
 X7 + X8 <> X6,
KIL2(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10):-
 X7 + X8 <> X9,
 X7 + X8 <> X10,
 X8 + X9 <> X1,
 X8 + X9 <> X2,
 X8 + X9 <> X3,
 X8 + X9 <> X4,
 X8 + X9 <> X5,
 X8 + X9 <> X6,
 X8 + X9 <> X7,
 X8 + X9 <> X10,
 X9 + X10 <> X1,
 X9 + X10 <> X2,
 X9 + X10 <> X3,
 X9 + X10 <> X4,
 X9 + X10 <> X5,
 X9 + X10 <> X6,
 X9 + X10 <> X7,
 X9 + X10 <> X8,
 X1 + X2 + X3 <> X4,
 X1 + X2 + X3 <> X5,
 X1 + X2 + X3 <> X6,
 X1 + X2 + X3 <> X7,
 X1 + X2 + X3 <> X8,
 X1 + X2 + X3 <> X9,
 X1 + X2 + X3 <> X10,
 X2 + X3 + X4 <> X1,
 X2 + X3 + X4 <> X5,
 X2 + X3 + X4 <> X6,
 X2 + X3 + X4 <> X7,
 X2 + X3 + X4 <> X8,
 X2 + X3 + X4 <> X9,
 X2 + X3 + X4 <> X10,
 X3 + X4 + X5 <> X1,
 X3 + X4 + X5 <> X2,
 X3 + X4 + X5 <> X6,
 X3 + X4 + X5 <> X7,
```

```
 X3 + X4 + X5 <> X8,
 X3 + X4 + X5 <> X9,
 X3 + X4 + X5 <> X10,
 X4 + X5 + X6 <> X1,
 X4 + X5 + X6 <> X2,
 X4 + X5 + X6 <> X3,
 X4 + X5 + X6 <> X7,
 X4 + X5 + X6 <> X8,
 X4 + X5 + X6 <> X9,
 X4 + X5 + X6 <> X10,
 X5 + X6 + X7 <> X1,
 X5 + X6 + X7 <> X2,
 X5 + X6 + X7 <> X3,
 X5 + X6 + X7 <> X4,
 X5 + X6 + X7 <> X8,
 X5 + X6 + X7 <> X9,
 X5 + X6 + X7 <> X10,
 X6 + X7 + X8 <> X1,
 X6 + X7 + X8 <> X2,
 X6 + X7 + X8 <> X3,
 X6 + X7 + X8 <> X4,
 X6 + X7 + X8 <> X5,
 X6 + X7 + X8 <> X9,
 X6 + X7 + X8 <> X10,
 X7 + X8 + X9 <> X1,
 X7 + X8 + X9 <> X2,
 X7 + X8 + X9 <> X3,
 X7 + X8 + X9 <> X4,
 X7 + X8 + X9 <> X5,
 X7 + X8 + X9 <> X6,
 X7 + X8 + X9 <> X10,
 X8 + X9 + X10 <> X1,
 X8 + X9 + X10 <> X2,
 X8 + X9 + X10 <> X3,
 X8 + X9 + X10 <> X4,
 X8 + X9 + X10 <> X5,
 X8 + X9 + X10 <> X6,
 X8 + X9 + X10 <> X7,
 X1 + X2 + X3 + X4 <> X5,
 X1 + X2 + X3 + X4 <> X6,
 X1 + X2 + X3 + X4 <> X7,
 X1 + X2 + X3 + X4 <> X8,
 X1 + X2 + X3 + X4 <> X9,
 X1 + X2 + X3 + X4 <> X10,
 X2 + X3 + X4 + X5 <> X1,
 X2 + X3 + X4 + X5 <> X6,
 X2 + X3 + X4 + X5 <> X7,
 X2 + X3 + X4 + X5 <> X8,
 X2 + X3 + X4 + X5 <> X9,
 X2 + X3 + X4 + X5 <> X10,
 X3 + X4 + X5 + X6 <> X1,
 X3 + X4 + X5 + X6 <> X2,
 X3 + X4 + X5 + X6 <> X7,
 X3 + X4 + X5 + X6 <> X8,
 X3 + X4 + X5 + X6 <> X9,
 X3 + X4 + X5 + X6 <> X10,
 X4 + X5 + X6 + X7 <> X1,
 X4 + X5 + X6 + X7 <> X2,
 X4 + X5 + X6 + X7 <> X3,
 X4 + X5 + X6 + X7 <> X8,
 X4 + X5 + X6 + X7 <> X9,
 X4 + X5 + X6 + X7 <> X10,
 X5 + X6 + X7 + X8 <> X1,
KIL3(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10):-
 X5 + X6 + X7 + X8 <> X2,
 X5 + X6 + X7 + X8 <> X3,
 X5 + X6 + X7 + X8 <> X4,
 X5 + X6 + X7 + X8 <> X9,
 X5 + X6 + X7 + X8 <> X10,
 X6 + X7 + X8 + X9 <> X1,
 X6 + X7 + X8 + X9 <> X2,
 X6 + X7 + X8 + X9 <> X3,
 X6 + X7 + X8 + X9 <> X4,
 X6 + X7 + X8 + X9 <> X5,
 X6 + X7 + X8 + X9 <> X10,
 X7 + X8 + X9 + X10 <> X1,
```

X7 + X8 + X9 + X10 <> X2,
X7 + X8 + X9 + X10 <> X3,
X7 + X8 + X9 + X10 <> X4,
X7 + X8 + X9 + X10 <> X5,
X7 + X8 + X9 + X10 <> X6,
X1 + X2 + X3 + X4 + X5 <> X6,
X1 + X2 + X3 + X4 + X5 <> X7,
X1 + X2 + X3 + X4 + X5 <> X8,
X1 + X2 + X3 + X4 + X5 <> X9,
X1 + X2 + X3 + X4 + X5 <> X10,
X2 + X3 + X4 + X5 + X6 <> X1,
X2 + X3 + X4 + X5 + X6 <> X7,
X2 + X3 + X4 + X5 + X6 <> X8,
X2 + X3 + X4 + X5 + X6 <> X9,
X2 + X3 + X4 + X5 + X6 <> X10,
X3 + X4 + X5 + X6 + X7 <> X1,
X3 + X4 + X5 + X6 + X7 <> X2,
X3 + X4 + X5 + X6 + X7 <> X8,
X3 + X4 + X5 + X6 + X7 <> X9,
X3 + X4 + X5 + X6 + X7 <> X10,
X4 + X5 + X6 + X7 + X8 <> X1,
X4 + X5 + X6 + X7 + X8 <> X2,
X4 + X5 + X6 + X7 + X8 <> X3,
X4 + X5 + X6 + X7 + X8 <> X9,
X4 + X5 + X6 + X7 + X8 <> X10,
X5 + X6 + X7 + X8 + X9 <> X1,
X5 + X6 + X7 + X8 + X9 <> X2,
X5 + X6 + X7 + X8 + X9 <> X3,
X5 + X6 + X7 + X8 + X9 <> X4,
X5 + X6 + X7 + X8 + X9 <> X10,
X6 + X7 + X8 + X9 + X10 <> X1,
X6 + X7 + X8 + X9 + X10 <> X2,
X6 + X7 + X8 + X9 + X10 <> X3,
X6 + X7 + X8 + X9 + X10 <> X4,
X6 + X7 + X8 + X9 + X10 <> X5,
X1 + X2 + X3 + X4 + X5 + X6 <> X7,
X1 + X2 + X3 + X4 + X5 + X6 <> X8,
X1 + X2 + X3 + X4 + X5 + X6 <> X9,
X1 + X2 + X3 + X4 + X5 + X6 <> X10,
X2 + X3 + X4 + X5 + X6 + X7 <> X1,
X2 + X3 + X4 + X5 + X6 + X7 <> X8,
X2 + X3 + X4 + X5 + X6 + X7 <> X9,
X2 + X3 + X4 + X5 + X6 + X7 <> X10,
X3 + X4 + X5 + X6 + X7 + X8 <> X1,
X3 + X4 + X5 + X6 + X7 + X8 <> X2,
X3 + X4 + X5 + X6 + X7 + X8 <> X9,
X3 + X4 + X5 + X6 + X7 + X8 <> X10,
X4 + X5 + X6 + X7 + X8 + X9 <> X1,
X4 + X5 + X6 + X7 + X8 + X9 <> X2,
X4 + X5 + X6 + X7 + X8 + X9 <> X3,
X4 + X5 + X6 + X7 + X8 + X9 <> X10,
X5 + X6 + X7 + X8 + X9 + X10 <> X1,
X5 + X6 + X7 + X8 + X9 + X10 <> X2,
X5 + X6 + X7 + X8 + X9 + X10 <> X3,
X5 + X6 + X7 + X8 + X9 + X10 <> X4,
X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X8,
X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X9,
X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X10,
X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X1,
X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X9,
X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X10,
X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X1,
X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X2,
X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X10,
X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1,
X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X2,
X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X3,
X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X9,
X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X10,
X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X1,
X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X10,
X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1,
X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X2,
X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X10,
X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1,
X1 + X2 + X3 <> X4 + X5,

```
 X1 + X2 + X3 <> X5 + X6,
 X1 + X2 + X3 <> X6 + X7,
 X1 + X2 + X3 <> X7 + X8,
 X1 + X2 + X3 <> X8 + X9,
 X1 + X2 + X3 <> X9 + X10,
 X2 + X3 + X4 <> X5 + X6,
 X2 + X3 + X4 <> X6 + X7,
 X2 + X3 + X4 <> X7 + X8,
 X2 + X3 + X4 <> X8 + X9,
 X2 + X3 + X4 <> X9 + X10,
 X3 + X4 + X5 <> X1 + X2,
KIL4(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10):-
 X3 + X4 + X5 <> X6 + X7,
 X3 + X4 + X5 <> X7 + X8,
 X3 + X4 + X5 <> X8 + X9,
 X3 + X4 + X5 <> X9 + X10,
 X4 + X5 + X6 <> X1 + X2,
 X4 + X5 + X6 <> X2 + X3,
 X4 + X5 + X6 <> X7 + X8,
 X4 + X5 + X6 <> X8 + X9,
 X4 + X5 + X6 <> X9 + X10,
 X5 + X6 + X7 <> X1 + X2,
 X5 + X6 + X7 <> X2 + X3,
 X5 + X6 + X7 <> X3 + X4,
 X5 + X6 + X7 <> X8 + X9,
 X5 + X6 + X7 <> X9 + X10,
 X6 + X7 + X8 <> X1 + X2,
 X6 + X7 + X8 <> X2 + X3,
 X6 + X7 + X8 <> X3 + X4,
 X6 + X7 + X8 <> X4 + X5,
 X6 + X7 + X8 <> X9 + X10,
 X7 + X8 + X9 <> X1 + X2,
 X7 + X8 + X9 <> X2 + X3,
 X7 + X8 + X9 <> X3 + X4,
 X7 + X8 + X9 <> X4 + X5,
 X7 + X8 + X9 <> X5 + X6,
 X8 + X9 + X10 <> X1 + X2,
 X8 + X9 + X10 <> X2 + X3,
 X8 + X9 + X10 <> X3 + X4,
 X8 + X9 + X10 <> X4 + X5,
 X8 + X9 + X10 <> X5 + X6,
 X8 + X9 + X10 <> X6 + X7,
 X1 + X2 + X3 + X4 <> X5 + X6,
 X1 + X2 + X3 + X4 <> X6 + X7,
 X1 + X2 + X3 + X4 <> X7 + X8,
 X1 + X2 + X3 + X4 <> X8 + X9,
 X1 + X2 + X3 + X4 <> X9 + X10,
 X2 + X3 + X4 + X5 <> X6 + X7,
 X2 + X3 + X4 + X5 <> X7 + X8,
 X2 + X3 + X4 + X5 <> X8 + X9,
 X2 + X3 + X4 + X5 <> X9 + X10,
 X3 + X4 + X5 + X6 <> X1 + X2,
 X3 + X4 + X5 + X6 <> X7 + X8,
 X3 + X4 + X5 + X6 <> X8 + X9,
 X3 + X4 + X5 + X6 <> X9 + X10,
 X4 + X5 + X6 + X7 <> X1 + X2,
 X4 + X5 + X6 + X7 <> X2 + X3,
 X4 + X5 + X6 + X7 <> X8 + X9,
 X4 + X5 + X6 + X7 <> X9 + X10,
 X5 + X6 + X7 + X8 <> X1 + X2,
 X5 + X6 + X7 + X8 <> X2 + X3,
 X5 + X6 + X7 + X8 <> X3 + X4,
 X5 + X6 + X7 + X8 <> X9 + X10,
 X6 + X7 + X8 + X9 <> X1 + X2,
 X6 + X7 + X8 + X9 <> X2 + X3,
 X6 + X7 + X8 + X9 <> X3 + X4,
 X6 + X7 + X8 + X9 <> X4 + X5,
 X7 + X8 + X9 + X10 <> X1 + X2,
 X7 + X8 + X9 + X10 <> X2 + X3,
 X7 + X8 + X9 + X10 <> X3 + X4,
 X7 + X8 + X9 + X10 <> X4 + X5,
 X7 + X8 + X9 + X10 <> X5 + X6,
 X1 + X2 + X3 + X4 + X5 <> X6 + X7,
 X1 + X2 + X3 + X4 + X5 <> X7 + X8,
 X1 + X2 + X3 + X4 + X5 <> X8 + X9,
 X1 + X2 + X3 + X4 + X5 <> X9 + X10,
```

```
 X2 + X3 + X4 + X5 + X6 <> X7 + X8,
 X2 + X3 + X4 + X5 + X6 <> X8 + X9,
 X2 + X3 + X4 + X5 + X6 <> X9 + X10,
 X3 + X4 + X5 + X6 + X7 <> X1 + X2,
 X3 + X4 + X5 + X6 + X7 <> X8 + X9,
 X3 + X4 + X5 + X6 + X7 <> X9 + X10,
 X4 + X5 + X6 + X7 + X8 <> X1 + X2,
 X4 + X5 + X6 + X7 + X8 <> X2 + X3,
 X4 + X5 + X6 + X7 + X8 <> X9 + X10,
 X5 + X6 + X7 + X8 + X9 <> X1 + X2,
 X5 + X6 + X7 + X8 + X9 <> X2 + X3,
 X5 + X6 + X7 + X8 + X9 <> X3 + X4,
 X6 + X7 + X8 + X9 + X10 <> X1 + X2,
 X6 + X7 + X8 + X9 + X10 <> X2 + X3,
 X6 + X7 + X8 + X9 + X10 <> X3 + X4,
 X6 + X7 + X8 + X9 + X10 <> X4 + X5,
 X1 + X2 + X3 + X4 + X5 + X6 <> X7 + X8,
 X1 + X2 + X3 + X4 + X5 + X6 <> X8 + X9,
 X1 + X2 + X3 + X4 + X5 + X6 <> X9 + X10,
 X2 + X3 + X4 + X5 + X6 + X7 <> X8 + X9,
 X2 + X3 + X4 + X5 + X6 + X7 <> X9 + X10,
 X3 + X4 + X5 + X6 + X7 + X8 <> X1 + X2,
 X3 + X4 + X5 + X6 + X7 + X8 <> X9 + X10,
 X4 + X5 + X6 + X7 + X8 + X9 <> X1 + X2,
 X4 + X5 + X6 + X7 + X8 + X9 <> X2 + X3,
 X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2,
 X5 + X6 + X7 + X8 + X9 + X10 <> X2 + X3,
 X5 + X6 + X7 + X8 + X9 + X10 <> X3 + X4,
 X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X8 + X9,
 X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X9 + X10,
 X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X9 + X10,
 X3 + X4 + X5 + X6 + X7 + X8 + X9 <> X1 + X2,
 X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2,
 X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X2 + X3,
 X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 <> X9 + X10,
KIL5(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10):-
 X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2,
 X1 + X2 + X3 + X4 <> X5 + X6 + X7,
 X1 + X2 + X3 + X4 <> X6 + X7 + X8,
 X1 + X2 + X3 + X4 <> X7 + X8 + X9,
 X1 + X2 + X3 + X4 <> X8 + X9 + X10,
 X2 + X3 + X4 + X5 <> X6 + X7 + X8,
 X2 + X3 + X4 + X5 <> X7 + X8 + X9,
 X2 + X3 + X4 + X5 <> X8 + X9 + X10,
 X3 + X4 + X5 + X6 <> X7 + X8 + X9,
 X3 + X4 + X5 + X6 <> X8 + X9 + X10,
 X4 + X5 + X6 + X7 <> X1 + X2 + X3,
 X4 + X5 + X6 + X7 <> X8 + X9 + X10,
 X5 + X6 + X7 + X8 <> X1 + X2 + X3,
 X5 + X6 + X7 + X8 <> X2 + X3 + X4,
 X6 + X7 + X8 + X9 <> X1 + X2 + X3,
 X6 + X7 + X8 + X9 <> X2 + X3 + X4,
 X6 + X7 + X8 + X9 <> X3 + X4 + X5,
 X7 + X8 + X9 + X10 <> X1 + X2 + X3,
 X7 + X8 + X9 + X10 <> X2 + X3 + X4,
 X7 + X8 + X9 + X10 <> X3 + X4 + X5,
 X7 + X8 + X9 + X10 <> X4 + X5 + X6,
 X1 + X2 + X3 + X4 + X5 <> X6 + X7 + X8,
 X1 + X2 + X3 + X4 + X5 <> X7 + X8 + X9,
 X1 + X2 + X3 + X4 + X5 <> X8 + X9 + X10,
 X2 + X3 + X4 + X5 + X6 <> X7 + X8 + X9,
 X2 + X3 + X4 + X5 + X6 <> X8 + X9 + X10,
 X3 + X4 + X5 + X6 + X7 <> X8 + X9 + X10,
 X4 + X5 + X6 + X7 + X8 <> X1 + X2 + X3,
 X5 + X6 + X7 + X8 + X9 <> X1 + X2 + X3,
 X5 + X6 + X7 + X8 + X9 <> X2 + X3 + X4,
 X6 + X7 + X8 + X9 + X10 <> X1 + X2 + X3,
 X6 + X7 + X8 + X9 + X10 <> X2 + X3 + X4,
 X6 + X7 + X8 + X9 + X10 <> X3 + X4 + X5,
 X1 + X2 + X3 + X4 + X5 + X6 <> X7 + X8 + X9,
 X1 + X2 + X3 + X4 + X5 + X6 <> X8 + X9 + X10,
 X2 + X3 + X4 + X5 + X6 + X7 <> X8 + X9 + X10,
 X4 + X5 + X6 + X7 + X8 + X9 <> X1 + X2 + X3,
 X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2 + X3,
 X5 + X6 + X7 + X8 + X9 + X10 <> X2 + X3 + X4,
 X1 + X2 + X3 + X4 + X5 + X6 + X7 <> X8 + X9 + X10,
```

X4 + X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2 + X3,
X1 + X2 + X3 + X4 + X5 <> X6 + X7 + X8 + X9,
X1 + X2 + X3 + X4 + X5 <> X7 + X8 + X9 + X10,
X2 + X3 + X4 + X5 + X6 <> X7 + X8 + X9 + X10,
X5 + X6 + X7 + X8 + X9 <> X1 + X2 + X3 + X4,
X6 + X7 + X8 + X9 + X10 <> X1 + X2 + X3 + X4,
X6 + X7 + X8 + X9 + X10 <> X2 + X3 + X4 + X5,
X1 + X2 + X3 + X4 + X5 + X6 <> X7 + X8 + X9 + X10,
X5 + X6 + X7 + X8 + X9 + X10 <> X1 + X2 + X3 + X4,
X1 + X2 <> X3 + X4,
X1 + X2 <> X4 + X5,
X1 + X2 <> X5 + X6,
X1 + X2 <> X6 + X7,
X1 + X2 <> X7 + X8,
X1 + X2 <> X8 + X9,
X1 + X2 <> X9 + X10,
X2 + X3 <> X4 + X5,
X2 + X3 <> X5 + X6,
X2 + X3 <> X6 + X7,
X2 + X3 <> X7 + X8,
X2 + X3 <> X8 + X9,
X2 + X3 <> X9 + X10,
X3 + X4 <> X5 + X6,
X3 + X4 <> X6 + X7,
X3 + X4 <> X7 + X8,
X3 + X4 <> X8 + X9,
X3 + X4 <> X9 + X10,
X4 + X5 <> X6 + X7,
X4 + X5 <> X7 + X8,
X4 + X5 <> X8 + X9,
X4 + X5 <> X9 + X10,
X5 + X6 <> X7 + X8,
X5 + X6 <> X8 + X9,
X5 + X6 <> X9 + X10,
X6 + X7 <> X8 + X9,
X6 + X7 <> X9 + X10,
X7 + X8 <> X9 + X10,
X1 + X2 + X3 <> X4 + X5 + X6,
X1 + X2 + X3 <> X5 + X6 + X7,
X1 + X2 + X3 <> X6 + X7 + X8,
X1 + X2 + X3 <> X7 + X8 + X9,
X1 + X2 + X3 <> X8 + X9 + X10,
X2 + X3 + X4 <> X5 + X6 + X7,
X2 + X3 + X4 <> X6 + X7 + X8,
X2 + X3 + X4 <> X7 + X8 + X9,
X2 + X3 + X4 <> X8 + X9 + X10,
X3 + X4 + X5 <> X6 + X7 + X8,
X3 + X4 + X5 <> X7 + X8 + X9,
X3 + X4 + X5 <> X8 + X9 + X10,
X4 + X5 + X6 <> X7 + X8 + X9,
X4 + X5 + X6 <> X8 + X9 + X10,
X5 + X6 + X7 <> X8 + X9 + X10,
X1 + X2 + X3 + X4 <> X5 + X6 + X7 + X8,
X1 + X2 + X3 + X4 <> X6 + X7 + X8 + X9,
X1 + X2 + X3 + X4 <> X7 + X8 + X9 + X10,
X2 + X3 + X4 + X5 <> X6 + X7 + X8 + X9,
X2 + X3 + X4 + X5 <> X7 + X8 + X9 + X10,
X3 + X4 + X5 + X6 <> X7 + X8 + X9 + X10,
X1 + X2 + X3 + X4 + X5 <> X6 + X7 + X8 + X9 + X10.

ДОДАТОК В. ПРОГРАМА ПОБУДОВИ ВСІХ ВАРІАНТІВ ЛІНІЙОК ГОЛОМБА ДЛЯ СХЕМ ВІДНОВЛЕННЯ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ З КІЛЬКІСТЮ $N = 24$ НА МОВІ VISUAL PROLOG

```
code=10000
domains
X=integer
C=integer
Z=integer
predicates
        kil(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil0(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil00(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil1(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil2(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil3(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil4(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil5(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil6(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil7(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil8(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil9(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil10(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil11(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil12(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil13(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil14(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil15(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil16(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil17(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil18(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil19(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil20(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil21(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil22(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil23(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil24(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil25(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil26(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil27(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil28(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil29(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil30(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil31(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil32(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil33(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil34(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil35(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil36(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil37(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil38(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil39(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil40(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil41(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil42(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil43(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil44(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil45(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil46(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil47(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil48(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil49(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil50(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil51(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil52(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
```

kil53(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil54(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil55(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil56(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil57(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil58(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil59(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil60(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil61(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil62(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil63(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil64(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil65(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil66(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil67(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil68(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil69(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil70(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil71(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil72(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil73(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil74(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil75(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil76(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil77(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil78(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil79(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil80(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil81(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil82(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil83(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil84(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil85(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil86(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil87(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil88(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil89(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil90(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil91(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil92(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil93(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil94(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil95(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil96(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil97(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil98(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil99(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil100(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil101(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil102(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil103(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil104(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil105(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil106(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil107(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil108(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil109(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil110(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil111(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil112(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil113(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil114(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil115(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil116(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil117(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil118(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil119(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil120(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil121(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil122(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil123(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil124(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil125(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil126(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil127(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
kil128(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).

```
        kil129(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil130(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil131(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil132(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil133(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil134(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil135(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil136(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil137(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil138(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil139(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil140(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil141(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil142(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil143(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil144(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil145(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil146(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil147(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil148(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil149(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil150(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil151(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        kil152(X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X).
        nondeterm cycle(X,X,X).
        nondeterm all(X).
        kil_all(X)
clauses
        cycle(N,N,Y):-N>=Y.
        cycle(N,X,Y):-Y<X,N2=X-1,cycle(N,N2,Y).
        all(X):-cycle(X,240,1).
kil_all(480):-
        cycle(P,480,360),nl,write("P=",P),nl,
        all(X1),all(X2),all(X3),all(X4),all(X5),
        all(X6),all(X7),all(X8),all(X9),all(X9),
        all(X10),all(X11),all(X12),all(X13),all(X14),
        all(X15),all(X16),all(X17),all(X18),all(X19),
        all(X20),all(X21),all(X22),all(X23),all(X24),
        X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11+X12+X13+X14+X15+X16+X17+X18+X19+X20+X21+X22+X23+X24<=P,
        kil(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),fail.
kil0(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
        kil1(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil2(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil3(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil4(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil5(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil6(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil7(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil8(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil9(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil10(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil11(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil12(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil13(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil14(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil15(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil16(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil17(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil18(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil19(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil20(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil21(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil22(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil23(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil24(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil25(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil26(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil27(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil28(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil29(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil30(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil31(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil32(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil33(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil34(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil35(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
```

kil36(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil37(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil38(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil39(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil40(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil41(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil42(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil43(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil44(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil45(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil46(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil47(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil48(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil49(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil50(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil51(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil52(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil53(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil54(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil55(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil56(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil57(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil58(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil59(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil60(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil61(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil62(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil63(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil64(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil65(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil66(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil67(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil68(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil69(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil70(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil71(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil72(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil73(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil74(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil75(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil76(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil77(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil78(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil79(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil80(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil81(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil82(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil83(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil84(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil85(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil86(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil87(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil88(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil89(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil90(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil91(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil92(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil93(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil94(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil95(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil96(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil97(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil98(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil99(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24).
kil00(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
kil100(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil101(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil102(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil103(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil104(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil105(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil106(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil107(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil108(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil109(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
kil110(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),

```
        kil111(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil112(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil113(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil114(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil115(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil116(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil117(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil118(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil119(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil120(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil121(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil122(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil123(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil124(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil125(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil126(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil127(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil128(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil129(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil130(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil131(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil132(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil133(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil134(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil135(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil136(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil137(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil138(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil139(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil140(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil141(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil142(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil143(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil144(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil145(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil146(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil147(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil148(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil149(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil150(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil151(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil152(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24).
        kil(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
        kil0(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24),
        kil00(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24).
kil1(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X1<>X2,
X1<>X3,
X1<>X4,
X1<>X5,
X1<>X6,
X1<>X7,
X1<>X8,
X1<>X9,
X1<>X10,
X1<>X11,
X1<>X12,
X1<>X13,
X1<>X14,
X1<>X15,
X1<>X16,
X1<>X17,
X1<>X18,
X1<>X19,
X1<>X20,
X1<>X21,
X1<>X22,
X1<>X23,
X1<>X24,
X2<>X3,
X2<>X4,
X2<>X5,
X2<>X6,
X2<>X7,
X2<>X8,
X2<>X9,
```

```
X2<>X10,
X2<>X11,
X2<>X12,
X2<>X13,
X2<>X14,
X2<>X15,
X2<>X16,
X2<>X17,
X2<>X18,
X2<>X19,
X2<>X20,
X2<>X21,
X2<>X22,
X2<>X23,
X2<>X24,
X3<>X4,
X3<>X5,
X3<>X6,
X3<>X7,
X3<>X8,
X3<>X9,
X3<>X10,
X3<>X11,
X3<>X12,
X3<>X13,
X3<>X14,
X3<>X15,
X3<>X16,
X3<>X17,
X3<>X18,
X3<>X19,
X3<>X20,
X3<>X21,
X3<>X22,
X3<>X23,
X3<>X24,
X4<>X5,
X4<>X6,
X4<>X7,
X4<>X8,
X4<>X9,
X4<>X10,
X4<>X11,
X4<>X12,
X4<>X13,
X4<>X14,
X4<>X15,
X4<>X16,
X4<>X17,
X4<>X18,
X4<>X19,
X4<>X20,
X4<>X21,
X4<>X22,
X4<>X23,
X4<>X24,
X5<>X6,
X5<>X7,
X5<>X8,
X5<>X9,
X5<>X10,
X5<>X11,
X5<>X12,
X5<>X13,
X5<>X14,
X5<>X15,
X5<>X16,
X5<>X17,
X5<>X18.
kil2(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X5<>X19,
X5<>X20,
X5<>X21,
X5<>X22,
X5<>X23,
X5<>X24,
```

X6<>X7,
X6<>X8,
X6<>X9,
X6<>X10,
X6<>X11,
X6<>X12,
X6<>X13,
X6<>X14,
X6<>X15,
X6<>X16,
X6<>X17,
X6<>X18,
X6<>X19,
X6<>X20,
X6<>X21,
X6<>X22,
X6<>X23,
X6<>X24,
X7<>X8,
X7<>X9,
X7<>X10,
X7<>X11,
X7<>X12,
X7<>X13,
X7<>X14,
X7<>X15,
X7<>X16,
X7<>X17,
X7<>X18,
X7<>X19,
X7<>X20,
X7<>X21,
X7<>X22,
X7<>X23,
X7<>X24,
X8<>X9,
X8<>X10,
X8<>X11,
X8<>X12,
X8<>X13,
X8<>X14,
X8<>X15,
X8<>X16,
X8<>X17,
X8<>X18,
X8<>X19,
X8<>X20,
X8<>X21,
X8<>X22,
X8<>X23,
X8<>X24,
X9<>X10,
X9<>X11,
X9<>X12,
X9<>X13,
X9<>X14,
X9<>X15,
X9<>X16,
X9<>X17,
X9<>X18,
X9<>X19,
X9<>X20,
X9<>X21,
X9<>X22,
X9<>X23,
X9<>X24,
X10<>X11,
X10<>X12,
X10<>X13,
X10<>X14,
X10<>X15,
X10<>X16,
X10<>X17,
X10<>X18,
X10<>X19,
X10<>X20,

```
X10<>X21,
X10<>X22,
X10<>X23,
X10<>X24,
X11<>X12,
X11<>X13,
X11<>X14,
X11<>X15,
X11<>X16,
X11<>X17,
X11<>X18,
X11<>X19,
X11<>X20,
X11<>X21,
X11<>X22,
X11<>X23,
X11<>X24.
kil3(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X12<>X13,
X12<>X14,
X12<>X15,
X12<>X16,
X12<>X17,
X12<>X18,
X12<>X19,
X12<>X20,
X12<>X21,
X12<>X22,
X12<>X23,
X12<>X24,
X13<>X14,
X13<>X15,
X13<>X16,
X13<>X17,
X13<>X18,
X13<>X19,
X13<>X20,
X13<>X21,
X13<>X22,
X13<>X23,
X13<>X24,
X14<>X15,
X14<>X16,
X14<>X17,
X14<>X18,
X14<>X19,
X14<>X20,
X14<>X21,
X14<>X22,
X14<>X23,
X14<>X24,
X15<>X16,
X15<>X17,
X15<>X18,
X15<>X19,
X15<>X20,
X15<>X21,
X15<>X22,
X15<>X23,
X15<>X24,
X16<>X17,
X16<>X18,
X16<>X19,
X16<>X20,
X16<>X21,
X16<>X22,
X16<>X23,
X16<>X24,
X17<>X18,
X17<>X19,
X17<>X20,
X17<>X21,
X17<>X22,
X17<>X23,
X17<>X24,
X18<>X19,
```

X18<>X20,
X18<>X21,
X18<>X22,
X18<>X23,
X18<>X24,
X19<>X20,
X19<>X21,
X19<>X22,
X19<>X23,
X19<>X24,
X20<>X21,
X20<>X22,
X20<>X23,
X20<>X24,
X21<>X22,
X21<>X23,
X21<>X24,
X22<>X23,
X22<>X24,
X23<>X24,
X1+X2<>X3,
X1+X2<>X4,
X1+X2<>X5,
X1+X2<>X6,
X1+X2<>X7,
X1+X2<>X8,
X1+X2<>X9,
X1+X2<>X10,
X1+X2<>X11,
X1+X2<>X12,
X1+X2<>X13,
X1+X2<>X14,
X1+X2<>X15,
X1+X2<>X16,
X1+X2<>X17,
X1+X2<>X18,
X1+X2<>X19,
X1+X2<>X20,
X1+X2<>X21,
X1+X2<>X22,
X1+X2<>X23.
і так далі.
kil150(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X5+X6+X7+X8+X9+X10<>X18+X19+X20+X21+X22+X23,
X5+X6+X7+X8+X9+X10<>X19+X20+X21+X22+X23+X24,
X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17,
X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18,
X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19,
X6+X7+X8+X9+X10+X11<>X15+X16+X17+X18+X19+X20,
X6+X7+X8+X9+X10+X11<>X16+X17+X18+X19+X20+X21,
X6+X7+X8+X9+X10+X11<>X17+X18+X19+X20+X21+X22,
X6+X7+X8+X9+X10+X11<>X18+X19+X20+X21+X22+X23,
X6+X7+X8+X9+X10+X11<>X19+X20+X21+X22+X23+X24,
X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18,
X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19,
X7+X8+X9+X10+X11+X12<>X15+X16+X17+X18+X19+X20,
X7+X8+X9+X10+X11+X12<>X16+X17+X18+X19+X20+X21,
X7+X8+X9+X10+X11+X12<>X17+X18+X19+X20+X21+X22,
X7+X8+X9+X10+X11+X12<>X18+X19+X20+X21+X22+X23,
X7+X8+X9+X10+X11+X12<>X19+X20+X21+X22+X23+X24,
X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19,
X8+X9+X10+X11+X12+X13<>X15+X16+X17+X18+X19+X20,
X8+X9+X10+X11+X12+X13<>X16+X17+X18+X19+X20+X21,
X8+X9+X10+X11+X12+X13<>X17+X18+X19+X20+X21+X22,
X8+X9+X10+X11+X12+X13<>X18+X19+X20+X21+X22+X23,
X8+X9+X10+X11+X12+X13<>X19+X20+X21+X22+X23+X24,
X9+X10+X11+X12+X13+X14<>X15+X16+X17+X18+X19+X20,
X9+X10+X11+X12+X13+X14<>X16+X17+X18+X19+X20+X21,
X9+X10+X11+X12+X13+X14<>X17+X18+X19+X20+X21+X22,
X9+X10+X11+X12+X13+X14<>X18+X19+X20+X21+X22+X23,
X9+X10+X11+X12+X13+X14<>X19+X20+X21+X22+X23+X24,
X10+X11+X12+X13+X14+X15<>X16+X17+X18+X19+X20+X21,
X10+X11+X12+X13+X14+X15<>X17+X18+X19+X20+X21+X22,
X10+X11+X12+X13+X14+X15<>X18+X19+X20+X21+X22+X23,
X10+X11+X12+X13+X14+X15<>X19+X20+X21+X22+X23+X24,
X11+X12+X13+X14+X15+X16<>X17+X18+X19+X20+X21+X22,

X11+X12+X13+X14+X15+X16<>X18+X19+X20+X21+X22+X23,
X11+X12+X13+X14+X15+X16<>X19+X20+X21+X22+X23+X24,
X12+X13+X14+X15+X16+X17<>X18+X19+X20+X21+X22+X23,
X12+X13+X14+X15+X16+X17<>X19+X20+X21+X22+X23+X24,
X13+X14+X15+X16+X17+X18<>X19+X20+X21+X22+X23+X24,
X1+X2+X3+X4+X5+X6+X7<>X8+X9+X10+X11+X12+X13+X14,
X1+X2+X3+X4+X5+X6+X7<>X9+X10+X11+X12+X13+X14+X15,
X1+X2+X3+X4+X5+X6+X7<>X10+X11+X12+X13+X14+X15+X16,
X1+X2+X3+X4+X5+X6+X7<>X11+X12+X13+X14+X15+X16+X17,
X1+X2+X3+X4+X5+X6+X7<>X12+X13+X14+X15+X16+X17+X18,
X1+X2+X3+X4+X5+X6+X7<>X13+X14+X15+X16+X17+X18+X19,
X1+X2+X3+X4+X5+X6+X7<>X14+X15+X16+X17+X18+X19+X20,
X1+X2+X3+X4+X5+X6+X7<>X15+X16+X17+X18+X19+X20+X21,
X1+X2+X3+X4+X5+X6+X7<>X16+X17+X18+X19+X20+X21+X22,
X1+X2+X3+X4+X5+X6+X7<>X17+X18+X19+X20+X21+X22+X23,
X1+X2+X3+X4+X5+X6+X7<>X18+X19+X20+X21+X22+X23+X24,
X2+X3+X4+X5+X6+X7+X8<>X9+X10+X11+X12+X13+X14+X15,
X2+X3+X4+X5+X6+X7+X8<>X10+X11+X12+X13+X14+X15+X16,
X2+X3+X4+X5+X6+X7+X8<>X11+X12+X13+X14+X15+X16+X17,
X2+X3+X4+X5+X6+X7+X8<>X12+X13+X14+X15+X16+X17+X18,
X2+X3+X4+X5+X6+X7+X8<>X13+X14+X15+X16+X17+X18+X19,
X2+X3+X4+X5+X6+X7+X8<>X14+X15+X16+X17+X18+X19+X20,
X2+X3+X4+X5+X6+X7+X8<>X15+X16+X17+X18+X19+X20+X21,
X2+X3+X4+X5+X6+X7+X8<>X16+X17+X18+X19+X20+X21+X22,
X2+X3+X4+X5+X6+X7+X8<>X17+X18+X19+X20+X21+X22+X23,
X2+X3+X4+X5+X6+X7+X8<>X18+X19+X20+X21+X22+X23+X24,
X3+X4+X5+X6+X7+X8+X9<>X10+X11+X12+X13+X14+X15+X16,
X3+X4+X5+X6+X7+X8+X9<>X11+X12+X13+X14+X15+X16+X17,
X3+X4+X5+X6+X7+X8+X9<>X12+X13+X14+X15+X16+X17+X18,
X3+X4+X5+X6+X7+X8+X9<>X13+X14+X15+X16+X17+X18+X19,
X3+X4+X5+X6+X7+X8+X9<>X14+X15+X16+X17+X18+X19+X20,
X3+X4+X5+X6+X7+X8+X9<>X15+X16+X17+X18+X19+X20+X21,
X3+X4+X5+X6+X7+X8+X9<>X16+X17+X18+X19+X20+X21+X22,
X3+X4+X5+X6+X7+X8+X9<>X17+X18+X19+X20+X21+X22+X23,
X3+X4+X5+X6+X7+X8+X9<>X18+X19+X20+X21+X22+X23+X24,
X4+X5+X6+X7+X8+X9+X10<>X11+X12+X13+X14+X15+X16+X17,
X4+X5+X6+X7+X8+X9+X10<>X12+X13+X14+X15+X16+X17+X18,
X4+X5+X6+X7+X8+X9+X10<>X13+X14+X15+X16+X17+X18+X19,
X4+X5+X6+X7+X8+X9+X10<>X14+X15+X16+X17+X18+X19+X20,
X4+X5+X6+X7+X8+X9+X10<>X15+X16+X17+X18+X19+X20+X21,
X4+X5+X6+X7+X8+X9+X10<>X16+X17+X18+X19+X20+X21+X22,
X4+X5+X6+X7+X8+X9+X10<>X17+X18+X19+X20+X21+X22+X23,
X4+X5+X6+X7+X8+X9+X10<>X18+X19+X20+X21+X22+X23+X24,
X5+X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17+X18,
X5+X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18+X19,
X5+X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19+X20,
X5+X6+X7+X8+X9+X10+X11<>X15+X16+X17+X18+X19+X20+X21,
X5+X6+X7+X8+X9+X10+X11<>X16+X17+X18+X19+X20+X21+X22,
X5+X6+X7+X8+X9+X10+X11<>X17+X18+X19+X20+X21+X22+X23,
X5+X6+X7+X8+X9+X10+X11<>X18+X19+X20+X21+X22+X23+X24,
X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19,
X6+X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19+X20,
X6+X7+X8+X9+X10+X11+X12<>X15+X16+X17+X18+X19+X20+X21,
X6+X7+X8+X9+X10+X11+X12<>X16+X17+X18+X19+X20+X21+X22,
X6+X7+X8+X9+X10+X11+X12<>X17+X18+X19+X20+X21+X22+X23,
X6+X7+X8+X9+X10+X11+X12<>X18+X19+X20+X21+X22+X23+X24,
X7+X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19+X20,
X7+X8+X9+X10+X11+X12+X13<>X15+X16+X17+X18+X19+X20+X21,
X7+X8+X9+X10+X11+X12+X13<>X16+X17+X18+X19+X20+X21+X22,
X7+X8+X9+X10+X11+X12+X13<>X17+X18+X19+X20+X21+X22+X23,
X7+X8+X9+X10+X11+X12+X13<>X18+X19+X20+X21+X22+X23+X24,
X8+X9+X10+X11+X12+X13+X14<>X15+X16+X17+X18+X19+X20+X21,
X8+X9+X10+X11+X12+X13+X14<>X16+X17+X18+X19+X20+X21+X22,
X8+X9+X10+X11+X12+X13+X14<>X17+X18+X19+X20+X21+X22+X23,
X8+X9+X10+X11+X12+X13+X14<>X18+X19+X20+X21+X22+X23+X24,
X9+X10+X11+X12+X13+X14+X15<>X16+X17+X18+X19+X20+X21+X22.
kil151(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X9+X10+X11+X12+X13+X14+X15<>X17+X18+X19+X20+X21+X22+X23,
X9+X10+X11+X12+X13+X14+X15<>X18+X19+X20+X21+X22+X23+X24,
X10+X11+X12+X13+X14+X15+X16<>X17+X18+X19+X20+X21+X22+X23,
X10+X11+X12+X13+X14+X15+X16<>X18+X19+X20+X21+X22+X23+X24,
X11+X12+X13+X14+X15+X16+X17<>X18+X19+X20+X21+X22+X23+X24,
X1+X2+X3+X4+X5+X6+X7+X8<>X9+X10+X11+X12+X13+X14+X15+X16,
X1+X2+X3+X4+X5+X6+X7+X8<>X10+X11+X12+X13+X14+X15+X16+X17,
X1+X2+X3+X4+X5+X6+X7+X8<>X11+X12+X13+X14+X15+X16+X17+X18,
X1+X2+X3+X4+X5+X6+X7+X8<>X12+X13+X14+X15+X16+X17+X18+X19,

X1+X2+X3+X4+X5+X6+X7+X8<>X13+X14+X15+X16+X17+X18+X19+X20,
X1+X2+X3+X4+X5+X6+X7+X8<>X14+X15+X16+X17+X18+X19+X20+X21,
X1+X2+X3+X4+X5+X6+X7+X8<>X15+X16+X17+X18+X19+X20+X21+X22,
X1+X2+X3+X4+X5+X6+X7+X8<>X16+X17+X18+X19+X20+X21+X22+X23,
X1+X2+X3+X4+X5+X6+X7+X8<>X17+X18+X19+X20+X21+X22+X23+X24,
X2+X3+X4+X5+X6+X7+X8+X9<>X10+X11+X12+X13+X14+X15+X16+X17,
X2+X3+X4+X5+X6+X7+X8+X9<>X11+X12+X13+X14+X15+X16+X17+X18,
X2+X3+X4+X5+X6+X7+X8+X9<>X12+X13+X14+X15+X16+X17+X18+X19,
X2+X3+X4+X5+X6+X7+X8+X9<>X13+X14+X15+X16+X17+X18+X19+X20,
X2+X3+X4+X5+X6+X7+X8+X9<>X14+X15+X16+X17+X18+X19+X20+X21,
X2+X3+X4+X5+X6+X7+X8+X9<>X15+X16+X17+X18+X19+X20+X21+X22,
X2+X3+X4+X5+X6+X7+X8+X9<>X16+X17+X18+X19+X20+X21+X22+X23,
X2+X3+X4+X5+X6+X7+X8+X9<>X17+X18+X19+X20+X21+X22+X23+X24,
X3+X4+X5+X6+X7+X8+X9+X10<>X11+X12+X13+X14+X15+X16+X17+X18,
X3+X4+X5+X6+X7+X8+X9+X10<>X12+X13+X14+X15+X16+X17+X18+X19,
X3+X4+X5+X6+X7+X8+X9+X10<>X13+X14+X15+X16+X17+X18+X19+X20,
X3+X4+X5+X6+X7+X8+X9+X10<>X14+X15+X16+X17+X18+X19+X20+X21,
X3+X4+X5+X6+X7+X8+X9+X10<>X15+X16+X17+X18+X19+X20+X21+X22,
X3+X4+X5+X6+X7+X8+X9+X10<>X16+X17+X18+X19+X20+X21+X22+X23,
X3+X4+X5+X6+X7+X8+X9+X10<>X17+X18+X19+X20+X21+X22+X23+X24,
X4+X5+X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17+X18+X19,
X4+X5+X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18+X19+X20,
X4+X5+X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19+X20+X21,
X4+X5+X6+X7+X8+X9+X10+X11<>X15+X16+X17+X18+X19+X20+X21+X22,
X4+X5+X6+X7+X8+X9+X10+X11<>X16+X17+X18+X19+X20+X21+X22+X23,
X4+X5+X6+X7+X8+X9+X10+X11<>X17+X18+X19+X20+X21+X22+X23+X24,
X5+X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19+X20,
X5+X6+X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19+X20+X21,
X5+X6+X7+X8+X9+X10+X11+X12<>X15+X16+X17+X18+X19+X20+X21+X22,
X5+X6+X7+X8+X9+X10+X11+X12<>X16+X17+X18+X19+X20+X21+X22+X23,
X5+X6+X7+X8+X9+X10+X11+X12<>X17+X18+X19+X20+X21+X22+X23+X24,
X6+X7+X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19+X20+X21,
X6+X7+X8+X9+X10+X11+X12+X13<>X15+X16+X17+X18+X19+X20+X21+X22,
X6+X7+X8+X9+X10+X11+X12+X13<>X16+X17+X18+X19+X20+X21+X22+X23,
X6+X7+X8+X9+X10+X11+X12+X13<>X17+X18+X19+X20+X21+X22+X23+X24,
X7+X8+X9+X10+X11+X12+X13+X14<>X15+X16+X17+X18+X19+X20+X21+X22,
X7+X8+X9+X10+X11+X12+X13+X14<>X16+X17+X18+X19+X20+X21+X22+X23,
X7+X8+X9+X10+X11+X12+X13+X14<>X17+X18+X19+X20+X21+X22+X23+X24,
X8+X9+X10+X11+X12+X13+X14+X15<>X16+X17+X18+X19+X20+X21+X22+X23,
X8+X9+X10+X11+X12+X13+X14+X15<>X17+X18+X19+X20+X21+X22+X23+X24,
X9+X10+X11+X12+X13+X14+X15+X16<>X17+X18+X19+X20+X21+X22+X23+X24,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X10+X11+X12+X13+X14+X15+X16+X17+X18,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X11+X12+X13+X14+X15+X16+X17+X18+X19,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X12+X13+X14+X15+X16+X17+X18+X19+X20,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X13+X14+X15+X16+X17+X18+X19+X20+X21,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X14+X15+X16+X17+X18+X19+X20+X21+X22,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X1+X2+X3+X4+X5+X6+X7+X8+X9<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X11+X12+X13+X14+X15+X16+X17+X18+X19,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X12+X13+X14+X15+X16+X17+X18+X19+X20,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X13+X14+X15+X16+X17+X18+X19+X20+X21,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X14+X15+X16+X17+X18+X19+X20+X21+X22,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X2+X3+X4+X5+X6+X7+X8+X9+X10<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X3+X4+X5+X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17+X18+X19+X20,
X3+X4+X5+X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18+X19+X20+X21,
X3+X4+X5+X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19+X20+X21+X22,
X3+X4+X5+X6+X7+X8+X9+X10+X11<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X3+X4+X5+X6+X7+X8+X9+X10+X11<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X4+X5+X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19+X20+X21,
X4+X5+X6+X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19+X20+X21+X22,
X4+X5+X6+X7+X8+X9+X10+X11+X12<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X4+X5+X6+X7+X8+X9+X10+X11+X12<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X5+X6+X7+X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19+X20+X21+X22,
X5+X6+X7+X8+X9+X10+X11+X12+X13<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X5+X6+X7+X8+X9+X10+X11+X12+X13<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X6+X7+X8+X9+X10+X11+X12+X13+X14<>X15+X16+X17+X18+X19+X20+X21+X22+X23,
X6+X7+X8+X9+X10+X11+X12+X13+X14<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X7+X8+X9+X10+X11+X12+X13+X14+X15<>X16+X17+X18+X19+X20+X21+X22+X23+X24,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10<>X11+X12+X13+X14+X15+X16+X17+X18+X19+X20,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10<>X12+X13+X14+X15+X16+X17+X18+X19+X20+X21,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10<>X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17+X18+X19+X20+X21,
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22,

```
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22,
X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X4+X5+X6+X7+X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X4+X5+X6+X7+X8+X9+X10+X11+X12+X13<>X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X5+X6+X7+X8+X9+X10+X11+X12+X13+X14<>X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X12+X13+X14+X15+X16+X17+X18+X19+X20+X21+X22,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22+X23,
X2+X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23+X24,
X3+X4+X5+X6+X7+X8+X9+X10+X11+X12+X13<>X14+X15+X16+X17+X18+X19+X20+X21+X22+X23+X24.
kil152(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16,X17,X18,X19,X20,X21,X22,X23,X24):-
X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11+X12<>X13+X14+X15+X16+X17+X18+X19+X20+X21+X22+X23+X24.
goal
kil_all(480).
```